28th International Symposium on Theoretical Aspects of Computer Science

STACS'11, March 10–12, 2011, Dortmund, Germany

Edited by

Thomas Schwentick
Christoph Dürr

LIPICS

SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

LIPIcs – Vol. 9 – STACS'11 www.dagstuhl.de/lipics

Editors

Thomas Schwentick	Christoph Dürr
TU Dortmund	Ecole Polytechnique
Dortmund	Paris
`thomas.schwentick@udo.edu`	`christoph.durr@lip6.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 978-3-939897-25-5

Published online and open access by
Schloss Dagstuhl – Leibniz-Center for Informatics GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany.

Publication date
March, 2011

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.

Digital Object Identifier: 10.4230/LIPIcs.STACS.2011.i

ISBN 978-3-939897-25-5 **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.

ISSN 1868-8969

www.dagstuhl.de/lipics

Foreword

The Symposium on Theoretical Aspects of Computer Science (STACS) is held alternately in France and in Germany. The conference of March 10-12, 2011, held in Dortmund, is the 28th in this 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), and Nancy (2010).

The interest in STACS has remained at a high level over the past years. The STACS 2011 call for papers led to 271 submissions from 45 countries. Each paper was assigned to three program committee members. The committee selected 54 papers during a two-week electronic meeting held in November. As co-chairs of the program committee, we would like to sincerely thank 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.

We would like to express our thanks to the three invited speakers, Susanne Albers, Véronique Cortier, and Georg Gottlob. Special thanks go to Andrei Voronkov for his EasyChair software (www.easychair.org). Moreover, we would like to warmly thank Matthias Niewerth for preparing the conference proceedings.

For the fourth time, this year's STACS proceedings are published in electronic form. A printed version was also available at the conference, by request. The electronic proceedings are available through several portals, and in particular through LIPIcs and HAL series. The proceedings of the Symposium, which are published electronically in the LIPIcs (Leibniz International Proceedings in Informatics) series, are available through Dagstuhl's website. The LIPIcs series provides an ISBN for the proceedings volume and manages the indexing issues. HAL is an electronic repository managed by several French research agencies. Both, HAL and LIPIcs, guarantee perennial, free and easy electronic access, while the authors will retain the rights over their work. The rights on the articles in the proceedings are kept with the authors and the papers are available freely, under a Creative Commons license (see www.stacs-conf.org/faq.html for more details).

STACS 2011 received funds from Deutsche Forschungsgemeinschaft (DFG) and from TU Dortmund. We thank them for their support!

February 2011 Thomas Schwentick and Christoph Dürr

28th International Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Conference Organization

Program Comittee

Dietmar Berwanger	ENS Cachan
Patrick Briest	Paderborn University
Christian Choffrut	LIAFA , Université Denis Diderot
Benjamin Doerr	MPI Saarbrücken
Christoph Dürr	Ecole Polytechnique (co-chair)
Leah Epstein	University of Haifa
Thomas Erlebach	University of Leicester
Michele Flammini	University of L'Aquila
Nicolas Hanusse	LaBRI Bordeaux
Markus Holzer	Gießen University
Dániel Marx	Tel Aviv University
Claire Mathieu	Brown University
Colin McDiarmid	Oxford University
Rolf Niedermeier	TU Berlin
Nicolas Ollinger	Aix-Marseille Universitié
Marco Pellegrini	CNR Pisa
Jean-Francois Raskin	Université Libre de Bruxelles
Thomas Schwentick	TU Dortmund (co-chair)
Jeffrey Shallit	University of Waterloo
Till Tantau	University of Lübeck
Sophie Tison	Université de Lille
Ronald de Wolf	CWI Amsterdam

Local Organization Comittee

Markus Brinkmann	TU Dortmund
Daniela Huvermann	TU Dortmund
Ahmet Kara	TU Dortmund
Katja Losemann	TU Dortmund
Wim Martens	TU Dortmund
Matthias Niewerth	TU Dortmund
Silvia Röse	TU Dortmund
Thomas Schwentick	TU Dortmund (chair)
Matthias Thimm	TU Dortmund
Thomas Zeume	TU Dortmund

28th International Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

External Reviewers

Marcel R. Ackermann
Krook Jin Ahn
Cyril Allauzen
Jean-Paul Allouche
Helmut Alt
Klaus Ambos-Spies
Pierre-Yves Angrand
Elliot Anshelevich
Adam Antonik
Pierre Arnoux
James Aspnes
Mohamed Faouzi Atig
John Augustine
Chen Avin
Eric Bach
Olivier Bailleux
Evangelos Bampas
Nikhil Bansal
Vince Bárány
Jérémie Barbay
Laurent Bartholdi
Luca Beccetti
Florent Becker
Eli Ben-Sasson
Alberto Bertoni
Nathalie Bertrand
Nadja Betzler
Olaf Beyersdorff
Laurent Bienvenu
Davide Bilò
Vittorio Bilò
Henrik Björklund
Francine Blanchet-Sadri
Michael Blondin
Achim Blumensath
Manuel Bodirsky
Andrej Bogdanov
Mikołaj Bojańczyk
Benedikt Bollig
Nicolas Bonichon
Henning Bordihn
Prosenjit K. Bose
Yacine Boufkhad
Pierre Boulet
Olivier Bournez
Andreas Brandstädt
Robert Bredereck
Jop Briët
Dimo Brockhoff
Véronique Bruyère
Andrei A. Bulatov
Jonathan F. Buss
Sergio Cabello
Olivier Carton
Didier Caucal
Julien Cervelle
Rohit Chadha
Amit Chakrabarti
Sourav Chakraborty
Tanmoy Chakraborty
Jérémie Chalopin
Pierre Charbit
Krishnendu Chatterjee
Yijia Chen
Yu-Fang Chen
Flavio Chierichetti
Marek Chrobak
Jacek Chrząszcz
Javier Cilleruelo
Ştefan Ciobâcă
Francisco Claude
Johanne Cohen
Amin Coja-Oghlan
Thomas Colcombet
Sébastien Collette
Colin Cooper
David Coudert
Bruno Courcelle
Maxime Crochemore
Aiswarya Cyriac
Flavio D'Alessandro
Anuj Dawar
Bastian Degener
Aldric Degorre
Bilel Derbel
Ziadi Djelloul
Shahar Dobzinski
Shlomi Dolev
Mike Domaratzki
Rob Downey
Laurent Doyen
Arnaud Durand
Martin Dyer
Rachid Echahed
Khaled Elbassioni
Michael Elberfeld
Amr Elmasry
Robert Elsässer
Matthias Englert
Lionel Eyraud-Dubois
Angelo Fanelli
Germain Faure
Lene M. Favrholdt
Sándor Fekete
Henning Fernau
Pascal Ferraro
Santiago Figueira
Emmanuel Filiot
Felix Fischer
Matthias Fitzi
Fedor V. Fomin
Enrico Formenti
Nikolaos Fountoulakis
Pierre Fraignaud
Blanchard François
Anna E. Frid
Tobias Friedrich
Henryk Fukś
Martin Fürer
Péter Gács
Pierre Ganty
Jane Gao
Yong Gao
Jugal Garg
Cyril Gavoille
Hugo Gimbert
Mathieu Giraud
Amy Glen
Leslie Ann Goldberg
Laure Gonnord
Laurent Gourvès
Serge Grigorieff
Luciano Gualà
Irène Guessarian
Pierre Guillon
Jiong Guo
Venkatesan Guruswami
Dan Gutfreund
Serge Haddad
Rolf Haenni
Matthew Hague

28th International Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Magnus M. Halldorsson
Sean Hallgren
Peter Hancock
Sariel Har-Peled
Sepp Hartung
Meng He
Brett Hemenway
Danny Hermelin
Markus Hinkelmann
Denis R. Hirschfeldt
Mika Hirvensalo
John Hitchcock
Martin Hoefer
Michael Hoffmann
Hendrik Jan Hoogeboom
Chien-Chung Huang
Falk Hüffner
Anna Huber
Paul Hunter
David Ilcinkas
Lucian Ilie
Csanád Imreh
Piotr Indyk
Kazuo Iwama
Sebastian Jakobi
Andreas Jakoby
Bart M. P. Jansen
Emmanuel Jeandel
Claude-Pierre Jeannerod
Mark Jerrum
Colette Johnen
Peter Jonsson
Hossein Jowhari
Jacques Justin
Frank Kammer
Christos A. Kapoutsis
Ahmet Kara
Jarkko J. Kari
Andreas Karrenbauer
Michael Kaufmann
Hans Kellerer
Barbara Kempkes
Iordanis Kerenidis
Yusik Kim
Daniel Kirsten
Ralf Klasing
Philip N. Klein
Peter Kling
Johannes Köbler
Christian Komusiewicz
Guy Kortsarz
Adrian Kosowski
Ioannis Koutis
Stefan Kratsch
Matthias Krause
Andreas Krebs
Steve Kremer
Stephan Kreutzer
Marcin Kubica
Antonín Kučera
Gregory Kucherov
Raghav Kulkarni
Dietrich Kuske
Martin Kutrib
Arnaud Labourel
James Laird
Stefan Langerman
Hubert Larchevêque
Machel Latteux
Massimo Lauria
François Le Gall
Christophe Lecoutre
Thierry Lecroq
Dimitri Leemans
Stephan Lemkens
Alex Leong
Martin Leucker
Debbie Leung
Peter Leupold
Florence Levé
Asaf Levin
Lionel Levine
Nutan Limaye
Maciej Liśkiewicz
Kamal Lodaya
Christof Löding
Markus Lohrey
Daniel Lokshtanov
Shachar Lovett
Hsueh-I Lu
Anna Lubiw
Dorel Lucanu
Bin Ma
Sofian Maabout
Frédéric Magniez
Ridha Mahjoub
Jean Mairesse
Andreas Malcher
Andreas Maletti
Guillaume Malod
Loris Marchal
Maurice Margenstern
Jean-Yves Marion
Nicolas Markey
Wim Martens
Barnaby Martin
Thierry Massart
Bodo Mathey
Jiri Matousek
Elvira Mayordomo
Frédéric Mazoit
Bertrand Mazure
Andrew McGregor
Jan Mehler
Giovanna Melideo
Stefan Mengel
Carlo Mereghetti
Julián Mestre
Yves Métivier
Mehdi Mhalla
Jakub Michaliszyn
Joseph S. Miller
Ioannis Millis
Nikola Milosavljevic
Neeldhara Misra
Rasmus Ejlers Møgelberg
Gianpiero Monaco
Mickaël Montassier
Nelma Moreira
Pat Morin
Rémi Morin
Luca Moscardelli
Dana Moshkovitz
Shay Mozes
Haiko Müller
Markus Müller-Olm
Ian Munro
Reza Naserasr
Alfredo Navarra
Alantha Newman
André Nichterlein
Joachim Niehren
André Nies
Damian Niwinski
Dirk Nowotka
Jan Obdržálek
Alexander Okhotin
Frédéric Olive
Svetlana Olonetsky
Jörg Olschewski

Krzysztof Onak
Friedrich Otto
Sang-il Oum
Konstantinos Panagiotou
Andrea Papst
Daniel Paulusma
Arnaud Pêcher
Xavier Pérez-Giménez
Sylvain Perifel
Seth Pettie
Thi Ha Duong Phan
Matthieu Picantin
Krzysztof Pietrzak
Peter Pietrzyk
Giovanni Pighizzini
Jean-Éric Pin
Ely Porat
Natacha Portier
Philippe Preux
Guido Proietti
Kirk Pruhs
Gabriele Puppis
Harald Räcke
Christophe Raffalli
Sergio Rajsbaum
Rajeev Raman
R. Ramanujam
Narad Rampersad
Mickaël Randour
Dror Rawitz
Igor Razgon
Mireille Régnier
Klaus Reinhardt
Rogério Reis
Rüdiger Reischuk
Gaétan Richard
Dan Roche
Heiko Röglin
Andrei E. Romashchenko
Peter Rossmanith
Kunihiko Sadakane
Chandan Saha
Lakhdar Sais
Hiroshi Sakamoto
Thomas Sauerwald
Saket Saurabh
Nitin Saxena
Nicolas Schabanel
Gilles Schaeffer
Guido Schäfer
Dominik Scheder
Christian Scheideler
Sven Schewe
Ildikó Schlotter
Heinz Schmitz
Sylvain Schmitz
Henning Schnoor
Ilka Schnoor
Pierre-Yves Schobbens
Lutz Schr"oder
Pascal Schweitzer
Stefan Schwwon
Hans Georg Seeding
Danny Segev
Frank Sehnke
Pranab Sen
Olivier Serre
Frédéric Servais
Mordechai Shalom
Yaoyun Shi
Amir Shpilka
Pedro V. Silva
Hnas Simon
Patrice Skéébold
Alexander Skopalik
Martin Skutella
William F. Smyth
Christian Sohler
Alexander Souza
Reto Spöhel
Anand Srivastav
Angelika Steger
Frank Stephan
Klaus Sutner
Wojciech Szpankowski
Siamak Taati
Michael Tautschnig
Kavitha Telikepalli
Jan Arne Telle
Véronique Terrier
Johannes Textor
Nguyen Kim Thang
Guillaume Theyssier
Dimitrios M. Thilikos
John Thistle
Stéphan Thomassé
Ioan Todinca
Marc Tommasi
Jacobo Torán
Szymon Toruńczyk
Corentin Travers
Kai Trojahner
Kostas Tsichlas
Dan Turetsky
Johannes Uhlmann
Vladimir V'yugin
René van Bevern
Rob van Stee
Pascal Vanier
Martin Vatshelle
Carmine Ventre
Nikolai Vereshchagin
Adrian R. Vetta
Antoine Vigneron
Goyal Vineet
Emanuele Viola
Mikhail V. Volkov
Heribert Vollmer
Fabian Wagner
Klaus Wagner
Markus Wagner
Magnus Wahlström
John Watrous
Oren Weimann
Steve Weis
Mathias Weller
Matthias Westermann
Thomas Wilke
Ryan Williams
Carola Winzen
Gerhard J. Woeginger
Prudence W. H. Wong
David Woodruff
Sergey Yekhanin
Raphael Yuster
Thomas Zeugmann
Thomas Zeume
Binhai Zhu
Sandra Zilles
Marius Zimand

Table of Contents

Foreword v
Conference Organization vii
External Reviewers ix
Table of Contents xiii

Invited Papers

Algorithms for Dynamic Speed Scaling
Susanne Albers 1

Structural Decomposition Methods and What They are Good For
Markus Aschinger, Conrad Drescher, Georg Gottlob, Peter Jeavons, and Evgenij Thorstensen 12

How to prove security of communication protocols? A discussion on the soundness of formal models w.r.t. computational ones.
Hubert Comon-Lundh and Véronique Cortier 29

Session 2A: Distributed and Fault–Tolerant Computing

Local dependency dynamic programming in the presence of memory faults
Saverio Caminiti, Irene Finocchi, and Emanuele G. Fusco 45

Tight bounds for rumor spreading in graphs of a given conductance
George Giakkoupis 57

Tight Bounds For Distributed MST Verification
Liah Kor, Amos Korman, and David Peleg 69

Session 2B: Data Words and Data Trees

Automata based verification over linearly ordered data domains
Luc Segoufin and Szymon Toruńczyk 81

Bottom-up automata on data trees and vertical XPath
Diego Figueira and Luc Segoufin 93

Data Monoids
M. Bojańczyk 105

Session 3A: Cuts and Flows

Minimum $s-t$ cut in undirected planar graphs when the source and the sink are close
Haim Kaplan and Yahav Nussbaum 117

Towards Duality of Multicommodity Multiroute Cuts and Flows: Multilevel Ball-Growing
Petr Kolman and Christian Scheideler 129

Session 3B: Computational Geometry

Compact Visibility Representation of Plane Graphs
Jiun-Jie Wang and Xin He 141

Telling convex from reflex allows to map a polygon
Jérémie Chalopin, Shantanu Das, Yann Disser, Matúš Mihalák, and Peter Widmayer 153

Session 4A: Kernelization

Cross-Composition: A New Technique for Kernelization Lower Bounds
Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch 165

Vertex Cover Kernelization Revisited: Upper and Lower Bounds for a Refined Parameter
Bart M. P. Jansen and Hans L. Bodlaender 177

Hitting forbidden minors: Approximation and Kernelization
Fedor V. Fomin, Daniel Lokshtanov, Neeldhara Misra, Geevarghese Philip, and Saket Saurabh 189

Session 4B: Morphisms, Words, Bio Computing

Self-Assembly of Arbitrary Shapes Using RNAse Enzymes: Meeting the Kolmogorov Bound with Small Scale Factor (extended abstract)
Erik D. Demaine, Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers 201

Weakly Unambiguous Morphisms
Dominik D. Freydenberger, Hossein Nevisi, and Daniel Reidenbach 213

On Minimal Sturmian Partial Words
F. Blanchet-Sadri and John Lensmire 225

Session 5A: SAT & CSP

Improving PPSZ for 3-SAT using Critical Variables
Timon Hertli, Robin A. Moser, and Dominik Scheder 237

The Complexity of Weighted Boolean #CSP Modulo k
Heng Guo, Sangxia Huang, Pinyan Lu, and Mingji Xia 249

The #CSP Dichotomy is Decidable
Martin Dyer and David Richerby 261

Session 5B: Cellular Automata

A speed-up of oblivious multi-head finite automata by cellular automata
Alex Borello, Gaétan Richard, and Véronique Terrier 273

Stochastic Cellular Automata Solve the Density Classification Problem with an Arbitrary Precision
Nazim Fatès 284

Probabilistic cellular automata, invariant measures, and perfect sampling
Ana Bušić, Jean Mairesse, and Irène Marcovici 296

Session 7A: Clustering and Learning

Analysis of Agglomerative Clustering
Marcel R. Ackermann, Johannes Blömer, Daniel Kuntze, and Christian Sohler ... 308

Measuring Learning Complexitywith Criteria Epitomizers
John Case and Timo Kötzing 320

On Parsimonious Explanations For 2-D Tree- and Linearly-Ordered Data
Howard Karloff, Flip Korn, Konstantin Makarychev, and Yuval Rabani 332

Session 7B: Logic

Unary negation
Balder ten Cate and Luc Segoufin 344

First-order Fragments with Successor over Infinite Words
Jakub Kallas, Manfred Kufleitner, and Alexander Lauser 356

The model checking problem for propositional intuitionistic logic with one variable is AC^1-complete
Martin Mundhenk and Felix Weiß 368

Session 8A: Scheduling 1

A Fast Algorithm for Multi-Machine Scheduling Problems with Jobs of Equal Processing Times
Alejandro López-Ortiz and Claude-Guy Quimper 380

Scheduling for Weighted Flow Time and Energy with Rejection Penalty
Sze-Hang Chan, Tak-Wah Lam, and Lap-Kei Lee 392

Session 8B: Graph Decomposition

Clique-width: When Hard Does Not Mean Impossible
Robert Ganian, Petr Hliněný, and Jan Obdržálek 404

From Pathwidth to Connected Pathwidth
Dariusz Dereniowski 416

Session 9A: Streaming

Polynomial Fitting of Data Streams with Applications to Codeword Testing
Andrew McGregor, Atri Rudra, and Steve Uurtamo 428

Spectral Sparsification in the Semi-Streaming Setting
Jonathan A. Kelner and Alex Levin 440

Session 9B: Recursion Theory

Solovay functions and K-triviality
Laurent Bienvenu, Wolfgang Merkle, and André Nies 452

Everywhere complex sequences and the probabilistic method
Andrey Yu. Rumyantsev 464

Session 10A: Scheduling 2

Online Scheduling with Interval Conflicts
Magnús M. Halldórsson, Boaz Patt-Shamir, and Dror Rawitz 472

Analysis of multi-stage open shop processing systems
Christian E.J. Eggermont, Alexander Schrijver, and Gerhard J. Woeginger 484

Session 10B: Regular Expressions

Graphs Encoded by Regular Expressions
Stefan Gulan 495

Extended Regular Expressions: Succinctness and Decidability
Dominik D. Freydenberger 507

Session 11A: Graph Algorithms

New Exact and Approximation Algorithms for the Star Packing Problem in Undirected Graphs
Maxim Babenko and Alexey Gusakov 519

Balanced Interval Coloring
Antonios Antoniadis, Falk Hüffner, Pascal Lenzner, Carsten Moldenhauer, and Alexander Souza 531

Session 11B: Algebra & Complexity

Symmetric Determinantal Representation of Weakly-Skew Circuits
Bruno Grenet, Erich L. Kaltofen, Pascal Koiran, and Natacha Portier 543

Randomness Efficient Testing of Sparse Black Box Identities of Unbounded Degree over the Reals
Markus Bläser and Christian Engels 555

Session 13A: Complexity of Graph & Group Problems

On Isomorphism Testing of Groups with Normal Hall Subgroups
Youming Qiao, Jayalal Sarma M.N., and Bangsheng Tang 567

Space Complexity of Perfect Matching in Bounded Genus Bipartite Graphs
Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N. V. Vinodchandran 579

The Recognition of Triangle Graphs
George B. Mertzios 591

Session 13B: Verification

Collapse Operation Increases Expressive Power of Deterministic Higher Order Pushdown Automata
Paweł Parys 603

Temporal Synthesis for Bounded Systems and Environments
Orna Kupferman, Yoad Lustig, Moshe Y. Vardi, and Mihalis Yannakakis 615

Linear temporal logic for regular cost functions
Denis Kuperberg 627

Session 14A: Geometry and Complexity

Bounds on the maximum multiplicity of some common geometric graphs
Adrian Dumitrescu, André Schulz, Adam Sheffer, and Csaba D. Tóth 637

On the computational complexity of Ham-Sandwich cuts, Helly sets, and related problems
Christian Knauer, Hans Raj Tiwary, and Daniel Werner 649

Session 14B: Query Complexity

Quantum query complexity of minor-closed graph properties
Andrew M. Childs and Robin Kothari 661

Three Query Locally Decodable Codes with Higher Correctness Require Exponential Length
Anna Gál and Andrew Mills 673

Index of Authors 685

Algorithms for Dynamic Speed Scaling *

Susanne Albers[1]

1 **Humboldt-Universität zu Berlin**
Department of Computer Science
Unter den Linden 6, 10099 Berlin, Germany
albers@informatik.hu-berlin.de
http://www2.informatik.hu-berlin.de/˜albers/

Abstract

Many modern microprocessors allow the speed/frequency to be set dynamically. The general goal is to execute a sequence of jobs on a variable-speed processor so as to minimize energy consumption. This paper surveys algorithmic results on dynamic speed scaling. We address settings where (1) jobs have strict deadlines and (2) job flow times are to be minimized.

2010 Mathematics Subject Classification 68Q25, 68W27, 68W40, 90B35

Keywords and phrases Competitive analysis, energy-efficiency, flow time, job deadline, offline algorithm, online algorithm, response time, scheduling, variable-speed processor.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.1

1 Introduction

Energy has become a scarce and expensive resource. There is a growing awareness in society that energy conservation and an efficient energy use are important issues. Power dissipation is also critical in computer systems. Electricity costs impose a substantial strain on the budget of data and computing centers. Google representatives report that if power consumption continues to grow, power costs can easily overtake hardware costs by a large margin [11]. In this context engineers are interested in low power rather than speed [30]. Moreover, energy-efficiency is a concern in portable and battery-operated devices that have proliferated in recent years. An effective energy use can considerably prolong the lifetime of a battery and hence the availability of a system.

A relatively new and very promising technique to save energy in computer systems is dynamic speed scaling. Chip manufacturers such as Intel, AMD and IBM produce microprocessors that can run at variable speed. Examples are the Intel SpeedStep and the AMD PowerNow. High speeds result in high performance but also high energy consumption. Lower speeds save energy but performance degrades. In dynamic speed scaling the processor speed is adjusted based on demand and performance constraints. The goal is to minimize energy consumption, while still providing a desired quality of service. The past years have witnessed considerable research interest in dynamic speed scaling. In this paper we survey results that have been developed in the algorithms community.

The well-known cube-root rule for CMOS devices states that the speed s of a device is proportional to the cube-root of the power or, equivalently, that power is proportional to s^3. The algorithms literature considers a generalization of this rule. If a processor runs at speed s, then the required power is $P(s) = s^\alpha$, where $\alpha > 1$ is a constant. Most algorithms papers

* Work supported by a Gottfried Wilhelm Leibniz Award of the German Research Foundation.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 1–11
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

consider this power function $P(s)$; some even work with more generalized convex functions. Obviously, energy consumption is power integrated over time.

Dynamic speed scaling leads to many challenging scheduling problems. The general goal is to execute a sequence of jobs on a variable-speed processor so as to optimize energy consumption and, possibly, a second objective. However, problems in speed scaling are more complex than those in standard scheduling: At any time a scheduler has to decide not only which job to execute but also which speed to use.

There has recently been considerable research interest in the design and analysis of dynamic speed scaling algorithms. The algorithms literature so far focuses mostly on two scenarios. In a first scenario jobs have strict deadlines and a scheduler has to construct feasible schedules minimizing energy consumption. We review important results for this setting in Section 2. In a second scenario jobs have no deadlines but their response times or flow times are to be minimized, measuring the responsiveness of a system. Here one has to combine energy minimization and flow time minimization. We present results for this scenario in Section 3.

For the various scenarios, two problem settings are of interest. In the offline setting all jobs to be processed are known in advance. Here one is interested in complexity results and fast polynomial time algorithms for computing optimal or nearly optimal schedules. In the online setting jobs arrive over time and an algorithm, at any time, has to make scheduling decisions without knowledge of any future jobs. Online strategies are evaluated using competitive analysis [33]. An online algorithm ALG is called c-competitive if for every input, i.e. for any job sequence, the objective function value (typically the energy consumption) of ALG is within c times the value of an optimal solution for that input.

2 Scheduling with deadlines

In a seminal paper, initiating the algorithmic study of speed scaling, Yao, Demers and Shenker [34] investigated a scheduling problem with strict job deadlines. It is by far the most extensively studied speed scaling problem.

Consider n jobs $J_1, \ldots J_n$ that have to be processed on a variable-speed processor. Each job J_i is specified by a release time r_i, a deadline d_i and a processing volume w_i. The release time and the deadline specify the time interval $[r_i, d_i]$ during which the job must be executed. The job may not be started before r_i and must be finished until d_i. The processing volume w_i is the amount of work that must be completed to finish the job. Intuitively w_i can be viewed as the total number of CPU cycles required by the job. The processing time of the job depends on the processor speed. If J_i is executed at speed s, then it takes w_i/s time units to finish the task. Preemption of jobs is allowed, i.e. the processing of a job may be stopped and resumed later. The goal is to construct a feasible schedule minimizing the total energy consumption

Yao, Demers and Shenker [34] make two simplifying assumptions. (1) There is no upper bound on the allowed processor speed. Hence a feasible schedule always exists. (2) The processor has a continuous spectrum of speeds. In the following we will present algorithms for this enhanced model. Then we will discuss how to relax the assumptions.

2.1 Basic algorithms

Yao et al. [34] developed elegant online and offline algorithms. We first present the offline strategy, which knows all the jobs along with their characteristics in advance. The algorithm is known as *YDS*, referring to the initials of the authors. Algorithm *YDS* computes a minimum energy schedule for a given job set in a series of rounds. In each round the algorithm identifies an interval of maximum density and computes a corresponding partial schedule for that interval. The *density* Δ_I of a time interval $I = [t, t']$ is the total processing volume to be completed in I divided by the length of I. More formally, let S_I be the set of jobs J_i that must be processed in I, i.e. that satisfy $[r_i, d_i] \subseteq I$. Then

$$\Delta_I = \frac{1}{|I|} \sum_{J_i \in S_I} w_i.$$

Intuitively, Δ_I is the minimum average speed necessary to complete all jobs that must be scheduled in I.

In each round, *YDS* determines the interval I of maximum density. In I the algorithm schedules the jobs of S_I at speed Δ_I according to *Earliest Deadline First (EDF)*. The *EDF* policy always processes the job having the earliest deadline among the available unfinished jobs. Then *YDS* removes the set S_I as well as the time interval I from the problem instance. More specifically, for any unscheduled job J_i with $d_i \in I$, the new deadline is set to $d_i := t$. For any unscheduled J_i with $r_i \in I$, the new release time is $r_i := t'$. Time interval I is discarded. A summary of *YDS* in pseudo-code is given below.

Algorithm YDS: Initially $\mathcal{J} := \{J_1, \ldots, J_n\}$. While $\mathcal{J} \neq \emptyset$, execute the following two steps. (1) Determine the interval I of maximum density. In I process the jobs of S_I at speed Δ_I according to *EDF*. (2) Set $\mathcal{J} := \mathcal{J} \setminus S_I$. Remove I from the time horizon and update the release times and deadlines of unscheduled jobs accordingly.

The algorithm computes optimal schedules.

▶ **Theorem 2.1.** [34] *For any job instance, YDS computes an optimal schedule minimizing the total energy consumption.*

Obviously, the running time of *YDS* is polynomial. When identifying intervals of maximum density, the algorithm only has to consider intervals whose boundaries are equal to the release times and deadlines of the jobs. Hence, a straightforward implementation of the algorithm has a running time of $O(n^3)$. Li et al. [29] showed that the time can be reduced to $O(n^2 \log n)$. Further improvements are possible if the job execution intervals form a tree structure [27].

In the online version of the problem, the jobs $J_1, \ldots, J_n$ arrive over time. A job J_i becomes known only at its arrival time r_i. At that time the deadline d_i and the processing volume w_i are also revealed. Recall that an online algorithm *ALG* is c-competitive if, for any job sequence, the total energy consumption of *ALG* is at most c times that of an optimal offline algorithm *OPT*.

Yao et al. [34] devised two online algorithms, called *Average Rate* and *Optimal Available*. For any incoming job J_i, *Average Rate* considers the *density* $\delta_i = w_i/(d_i - r_i)$, which is the minimum average speed necessary to complete the job in time if no other jobs were present. At any time t the speed $s(t)$ is set to the accumulated density of jobs active at time t. A job J_i is *active at time t* if $t \in [r_i, d_i]$. Available jobs are scheduled according to the *EDF* policy.

Algorithm Average Rate: At any time t the processor uses a speed of $s(t) = \sum_{J_i : t \in [r_i, d_i]} \delta_i$. Available unfinished jobs are scheduled using *EDF*.

Yao et al. [34] proved an upper bound on the competitiveness.

▶ **Theorem 2.2.** [34] *The competitive ratio of Average Rate is at most $2^{\alpha-1}\alpha^\alpha$, for any $\alpha \geq 2$.*

Bansal et al. [3] demonstrated that the analysis is essentially tight by giving a nearly matching lower bound.

▶ **Theorem 2.3.** [3] *The competitive ratio of Average Rate is at least* $((2-\delta)\alpha)^\alpha/2$*, where* δ *is a function of* α *that approaches zero as* α *tends to infinity.*

The second strategy *Optimal Available* is computationally more expensive than *Average Rate*. It always computes an optimal schedule for the currently available work load. This can be done using *YDS*.

Algorithm Optimal Available: Whenever a new job arrives, compute an optimal schedule for the currently available unfinished jobs.

Bansal, Kimbrel and Pruhs [7] analyzed the above algorithm and proved the following result.

▶ **Theorem 2.4.** [7] *The competitive ratio of Optimal Available is exactly* α^α*.*

The above theorem implies that in terms of competitiveness, *Optimal Available* is better than *Average Rate*. Bansal et al. [7] also developed a new online algorithm, called *BKP* according to the initials of the authors, that approximates the optimal speeds of *YDS* by considering interval densities. For times t, t_1 and t_2 with $t_1 < t \leq t_2$, let $w(t, t_1, t_2)$ be the total processing volume of jobs that are active at time t, have a release time of at least t_1 and a deadline of at most t_2.

Algorithm BKP: At any time t use a speed of

$$s(t) = \max_{t'>t} \frac{w(t, et-(e-1)t', t')}{t'-t}.$$

Available unfinished jobs are processed using *EDF*.

▶ **Theorem 2.5.** [7] *Algorithm BKP achieves a competitive ratio of* $2(\frac{\alpha}{\alpha-1})^\alpha e^\alpha$*.*

For large values of α, the competitiveness of *BKP* is better than that of *Optimal Available*. Bansal et al. [6] gave an algorithm that achieves further improved bounds for small values of α, i.e. $\alpha = 2$ and $\alpha = 3$.

All the above online algorithms attain constant competitive ratios that depend on α but no other problem parameter. The dependence on α is exponential. For small values of α, which occur in practice, the competitive ratios are reasonably small. Moreover, results by Bansal et al. [6, 7] imply that the exponential dependence on α is inherent to the problem.

▶ **Theorem 2.6.** [6] *Any deterministic online algorithm has a competitiveness of at least* $e^{\alpha-1}/\alpha$*.*

Even randomized online algorithms have a competitive ratio of $\Omega((4/3)^\alpha)$, see [7]. An interesting open problem is to determine the best competitiveness that can be achieved by online algorithms.

2.2 Speed-bounded processors

The algorithms presented in the last section are designed for processors having available a continuous, unbounded spectrum of speeds. However, in practice a processor is equipped with only a finite set of discrete speed levels $s_1 < s_2 < \ldots < s_d$. The offline algorithm *YDS* can be modified easily to handle feasible job instances, i.e. inputs for which feasible schedules exist using the restricted set of speeds. Feasibility can be checked easily by always using the maximum speed s_d and scheduling available jobs according to the *EDF* policy. Given a feasible job instance the modification of *YDS* is as follows. We first construct the schedule according to *YDS*. For each identified interval I of maximum density we approximate the desired speed Δ_I by the two adjacent speed levels s_k and s_{k+1}, such that $s_k < \Delta_I < s_{k+1}$. Speed s_{k+1} is used first for some δ time units and s_k is used for the last $|I| - \delta$ time units in I, where δ is chosen such that the total work completed in I is equal to the original amount of $|I|\Delta_I$. An algorithm with an improved running time of $O(dn \log n)$ was presented by Li and Yao [28].

If the given job instance is not feasible, it is impossible to complete all the jobs. Here the goal is to design algorithms that achieve good *throughput*, which is the total processing volume of jobs finished by their deadline, and at the same time optimize energy consumption. Papers [4, 15] present algorithms that even work online. At any time the strategies maintain a pool of jobs they intend to complete. Newly arriving jobs may be admitted to this pool. If the pool contains too large a processing volume, jobs are expelled such that the throughput is not diminished significantly. The algorithm with the best competitiveness currently known is due to Bansal et al. [4]. The algorithm, called *Slow-D*, is 4-competitive in terms of throughput and constant competitive with respect to energy consumption. We describe the strategy.

Slow-D assumes that the processor has a continuous speed spectrum that is upper bounded by a maximum speed $s_{\max}$. The algorithm always keeps track of the speeds that *Optimal Available* would use for the workload currently available. At any time t *Slow-D* uses the speed that *Optimal Available* would set at time t provided that this speed does not exceed $s_{\max}$; otherwise *Slow-D* uses $s_{\max}$. The algorithm also considers scheduling times that are critical in terms of speed. For any t, down-time(t) is the latest time $t' \geq t$ in the future schedule such that the speed of *Optimal Available* is at least $s_{\max}$. If no such time exists, down-time(t) is set to the most recent time when $s_{\max}$ was used or to 0 if this has never been the case. Using this definition, jobs are labeled as *urgent* or *slack*. These labels may change over time. A job J_i is called t-urgent if $d_i \leq$ down-time(t); otherwise it is called t-slack. Additionally, *Slow-D* maintains two queues Q_{work} and Q_{wait} of jobs it intends to process. The status of Q_{work} defines *urgent periods*. An urgent period starts at the release time r_i of a job J_i if Q_{work} contained no urgent job right before r_i and J_i is an urgent job admitted to Q_{work} at time r_i. An urgent period ends at time t if Q_{work} contains no more t-urgent jobs. *Slow-D* works as follows.

Algorithm Slow-D: JOB ARRIVAL: A job J_i arriving at time r_i is admitted to Q_{work} if it is r_i-slack or if J_i and all the remaining work of r_i-urgent jobs in Q_{work} can be completed using $s_{\max}$. Otherwise J_i is appended to Q_{wait}.
JOB INTERRUPT: Whenever a job J_i in Q_{wait} reaches its last starting time $t = d_i - w_i/s_{\max}$, it raises an interrupt. At this time the algorithm is in an urgent period. Let J_k be the last job transfered from Q_{wait} to Q_{work} in the current period. If no such job exists, let J_k be a dummy job of processing volume zero transfered just before the current period started. Let W be the total original work of jobs ever admitted to Q_{work} that have become urgent after J_k was transfered to Q_{work}. If $w_i > 2(w_k + W)$, then remove all t-urgent jobs from Q_{work} and admit J_i; otherwise discard J_i.
JOB COMPLETION: Whenever a job is completed, it is removed from Q_{work}.

Bansal et al. [4] analyzed the above algorithm and proved the following result.

▶ **Theorem 2.7.** [4] *Slow-D is 4-competitive with respect to throughput and* $(\alpha^\alpha + \alpha^2 4^\alpha)$*-competitive with respect to energy.*

Interestingly, the competitiveness of 4 is best possible, even if energy is ignored, see [12].

2.3 Problem extensions

We consider further extensions of the classical deadline-based scheduling setting.

Sleep states: Irani et al. [22] investigate an extended scenario where a variable-speed processor may be transitioned into a sleep state. In the sleep state, the energy consumption is 0 while in the active state even at speed 0 some non-negative amount of energy is consumed. Hence [22] combines speed scaling with power-down mechanisms. In the standard setting without sleep state, algorithms tend to use low speed levels subject to release time and deadline constraints. In contrast, in the setting with sleep state it can be beneficial to speed up a job so as to generate idle times in which the processor can be transitioned to the sleep mode. Irani et al. [22] develop online and offline algorithms for this extended setting. For the online setting an algorithm with an improved competitiveness was presented by Han et al. [21]; their strategy achieves a competitiveness of $\alpha^\alpha + 2$. Baptiste [9], Baptiste et al. [10] and Demaine et al. [18] also study scheduling problems where a processor may be set asleep, albeit in a setting without speed scaling.

Parallel processors: The results presented so far address single-processor architectures. However, energy consumption is also a major concern in multi-processor environments. Consider a setting with m identical parallel processors. As usual the processing of a jobs may be preempted at any time. We distinguish two problem variants depending on whether or not *job migration* is allowed. If job migration is feasible, then whenever a job is preempted it may be moved to another processor. In some applications job migration can be an expensive or undesirable operation, and thus might be infeasible. In any case the goal is to minimize the total energy consumption on all the processors. Bingham and Greenstreet [13] showed that if job migration is allowed, the offline problem is polynomially solvable. However the corresponding algorithm relies on linear programming and, as the authors mention, the complexity of the algorithm might be too high for most practical applications.

Albers et al. [2] assume that job migration is not allowed. They show that the offline problem is NP-hard, even for unit-size jobs. Albers et al. [2] then develop polynomial time offline algorithms that achieve constant factor approximations, i.e. for any input the consumed energy is within a constant factor of the true optimum. They also devise online algorithms attaining constant competitive ratios. Greiner et al. [19] gave a strategy that converts a c-approximation algorithm for a single processor into a randomized cB_α-approximation algorithm for multiple processors. Here B_α is the α-th Bell number. A corresponding statement holds for online algorithms.

Lam et al. [24] study deadline-based scheduling on two speed-bounded processors. They present a strategy that is constant competitive in terms of throughput maximization and energy minimization.

3 Minimizing flow times

A classical objective in scheduling is the minimization of response times. A user releasing a task to a system would like to receive feedback, say the result of a computation, as quickly as possible. User satisfaction often depends on how fast a device reacts. Unfortunately, response time minimization and energy minimization are contradicting objectives. To achieve fast response times a system must usually use high processor speeds, which lead to high energy consumption. On the other hand, to save energy low speeds should be used, which result in high response times. Hence one has to find ways to integrate both objectives.

Consider n jobs $J_1, \ldots, J_n$ that have to be scheduled on a variable-speed processor. Each job J_i is specified by a release time r_i and a processing volume w_i. When a job arrives, its processing volume is known. Preemption of jobs is allowed. In the scheduling literature, response time is referred to as *flow time*. The flow time f_i of a job J_i is the length of the time interval between release time and completion time of the job. We seek schedules minimizing the total flow time $\sum_{i=1}^{n} f_i$.

3.1 Energy plus flow

Albers and Fujiwara [1] proposed the following approach to integrate energy and flow time minimization. They consider a combined objective function that simply adds the two costs. Let E denote the energy consumption of a schedule. We wish to minimize $g = E + \sum_{i=1}^{n} f_i$. By multiplying either the energy or the flow time by a scalar, we can also consider a weighted combination of the two costs, expressing the relative value of the two terms in the total cost. Albers and Fujiwara [1] concentrate on the setting where all jobs have the same processing volume. By scaling, one can assume that all jobs have unit-size. They show that optimal offline schedules can be constructed in polynomial time using a dynamic programming approach.

Most of [1] is concerned with the online setting where jobs arrive over time. Albers and Fujiwara present a simple online strategy that processes jobs in batches and achieves a constant competitive ratio. Batched processing allows one to make scheduling decisions, which are computationally expensive, only every once in a while. This is certainly an advantage in low-power computing environments. Nonetheless, Albers and Fujiwara conjectured that the following algorithm achieves a better performance with respect to the minimization of g: At any time, if there are ℓ active jobs, use speed $\sqrt[\alpha]{\ell}$. A job is active if it has been released but is still unfinished. Intuitively, this is a reasonable strategy because, in each time unit, the incurred energy of $(\sqrt[\alpha]{\ell})^{\alpha} = \ell$ is equal to the additional flow time accumulated by the ℓ jobs during that time unit. Hence, both energy and flow time contribute the same value to the objective function. The algorithm and variants thereof have been the subject of extensive analyses [4, 5, 8, 26], not only for unit-size jobs but also for arbitrary size jobs. Moreover, unweighted and weighted flow times have been considered.

The currently best result is due to Bansal et al. [5]. They modify the above algorithm slightly by using a speed of $\sqrt[\alpha]{\ell + 1}$ whenever ℓ jobs are active. Inspired by a paper of Lam et al. [26] they apply the *Shortest Remaining Processing Time (SRPT)* policy to the available jobs. More precisely, at any time among the active jobs, the one with the least remaining work is scheduled.

Algorithm Job Count: At any time if there are $\ell \geq 1$ active jobs, use speed $\sqrt[\alpha]{\ell + 1}$. If no job is available, use speed 0. Always schedule the job with the least remaining unfinished work.

▶ **Theorem 3.1.** [5] *Job Count is 3-competitive for arbitrary size jobs.*

The above result even holds for a very general class of convex power functions. Bansal et al. [5, 8] study a generalized setting where each job J_i has a weight β_i associated with it and in objective function g the total flow time is replaced by the weighted flow time $\sum_{i=1}^{n} \beta_i f_i$. The proposed algorithms rely on the *Highest Density First (HDF)* policy, i.e. at any time among the available unfinished jobs the one with the highest *density* is processed. The density of a job J_i is the ratio β_i/w_i of its weight to its work. Bansal et al. [8] introduced a relaxed objective function consisting of energy plus the fractional weighted flow time of the jobs. In the fractional weighted flow time measure, at any time a job contributes its weight times the percentage of unfinished work to the objective. In their first paper Bansal et al. [8] gave a constant competitive online algorithm for minimizing energy plus fractional weighted flow. An algorithm achieving a small constant competitive ratio of 2 was shown in the second paper [5]. This algorithm always applies *HDF* for job selection and sets the processor power equal to the total fractional weight of the unfinished jobs. A constant competitive algorithm for the original objective function of energy plus (integral) weighted flow was shown in [8].

Bansal et al. [4] and Lam et al. [26] propose algorithms for the setting that there is an upper bound on the maximum processor speed. All the results mentioned so far assume that when a job arrives, its processing volume is known. Articles [16, 26] investigate the harder case that this information is not available.

3.2 Problem extensions and modifications

Sleep states: Lam et al. [23] study an extended setting where a variable-speed processor is equipped with one or several sleep states. The processing time of incoming jobs may or may not be known. The authors devise online algorithms achieving constant competitive ratios for minimizing energy plus flow.

Parallel processors: Lam et al. [25] and Gupta et al. [20] investigate scenarios with m parallel processors. Both articles assume that job migration is not allowed. For identical processors Lam et al. [25] present a constant competitive online algorithm for minimizing energy plus flow. The performance ratio even holds against migratory offline schedules. The corresponding algorithm classifies jobs according to their processing volumes and was originally proposed by Albers et al. [2]. Gupta et al. [20] consider heterogeneous processors and study the effect of resource augmentation: If an offline algorithm can run a processor at speed s and power $P(s)$, then an online algorithm is able to run the processor at speed $(1+\epsilon)s$ and power $P(s)$, for any given $\epsilon > 0$. Gupta et al. present an online algorithm that is *scalable* for minimizing energy plus weighted flow. Here scalable means that the online cost is upper bounded by $O(f(\epsilon))$ time the optimum cost, where f is a polynomial function of small degree. Again the result holds for a very general class of power functions. If the power functions of all the processors are of the form $P_i(s) = s^{\alpha_i}$, $1 \leq i \leq m$, Gupta et al. show a $O(\alpha^2)$-competitive algorithm, where $\alpha = \max_i \alpha_i$. Hence resource augmentation is not needed. Chan et al. [17] investigate parallel processor scheduling assuming that jobs have varying degrees of parallelizability and their processing times are initially unknown.

Limited energy: Pruhs et al. [31] consider another approach to integrate energy and flow time minimization. More specifically they study a problem where a fixed energy volume E is given and the goal is to minimize the total flow time of the jobs. Pruhs et al. [31] assume that all jobs have unit-size. They consider the offline scenario and show that optimal schedules can be computed in polynomial time. Bunde [14] extends the result to parallel processor environments and gives an arbitrarily-good approximation for scheduling unit-size jobs. He also shows that the optimal flow time value cannot be exactly computed on a machine supporting exact real arithmetic, including the extraction of roots. We remark that in the framework with a limited energy volume it is hard to construct good online algorithms. If future jobs are unknown, it is unclear how much energy to invest for the currently available tasks.

4 Conclusions

In this paper we have surveyed algorithmic results on dynamic speed scaling, focusing on settings with strict job deadlines and on the minimization of job flow times. Various papers have also addressed other scenarios. A basic objective function in scheduling is makespan minimization, i.e. the minimization of the point in time when the entire schedule ends. Bunde [9] develops algorithms for single and multi-processor environments. Pruhs et al. [32] consider tasks having precedence constraints defined between them. They devise algorithms for parallel processors given a fixed energy volume. In summary, practical applications motivate the investigation of many further settings and we expect that dynamic speed scaling continues to be an active area of research.

References

1 S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3, 2007.

2 S. Albers, F. Müller and S. Schmelzer. Speed scaling on parallel processors. *Proc. 19th ACM Symposium on Parallelism in Algorithms and Architectures*, 289–298, 2007.

3 N. Bansal, D.P. Bunde, H.-L. Chan and K. Pruhs. Average rate speed scaling. *Proc. 8th Latin American Symposium on Theoretical Informatics,* Springer LNCS 4957, 240–251, 2008.

4 N. Bansal, H.-L. Chan, T.-W. Lam and K.-L. Lee. Scheduling for speed bounded processors. *Proc. 35th International Colloquium on Automata, Languages and Programming*, Springer LNCS 5125, 409–420, 2008.

5 N. Bansal, H.-L. Chan and K. Pruhs. Speed scaling with an arbitrary power function. *Proc. 20th ACM-SIAM Symposium on Discrete Algorithm*, 693–701, 2009.

6 N. Bansal, H.-L. Chan, K. Pruhs and D. Katz. Improved bounds for speed scaling in devices obeying the cube-root rule. *Proc. 36th International Colloqium on Automata, Languages and Programming*, Springer LNCS 5555, 144–155, 2009.

7 N. Bansal, T. Kimbrel and K. Pruhs. Speed scaling to manage energy and temperature. *Journal of the ACM,* 54, 2007.

8 N. Bansal, K. Pruhs and C. Stein. Speed scaling for weighted flow time. *SIAM Journal on Computing*, 39:1294–1308, 2009.

9 P. Baptiste. Scheduling unit tasks to minimize the number of idle periods: a polynomial time algorithm for offline dynamic power management. *Proc. 17th Annual ACM-SIAM Symposium on Discrete Algorithms*, 364–367, 2006.

10 P. Baptiste, M. Chrobak and C. Dürr. Polynomial time algorithms for minimum energy scheduling. *Proc. 15th Annual European Symposium on Algorithms*, Springer LNCS 4698, 136–150, 2007.

11 L.A. Barroso. The price of performance. *ACM Queue*, 3, 2005.

12 S.K. Baruah, G. Koren, B. Mishra, A. Raghunathan, L.E. Rosier and D. Shasha. On-line scheduling in the presence of overload. *Proc. 32nd Annual Symposium on Foundations of Computer Science*, 100–110, 1991.

13 B.D. Bingham and M.R. Greenstreet. Energy optimal scheduling on multiprocessors with migration. *Proc. IEEE International Symposium on Parallel and Distributed Processing with Applications*, 143–152, 2008.

14 D.P. Bunde. Power-aware scheduling for makespan and flow. *Journal of Scheduling*, 12:489–500, 2009.

15 H.-L. Chan, W.-T. Chan, T.-W. Lam, K.-L. Lee, K.-S. Mak and P.W.H. Wong. Optimizing throughput and energy in online deadline scheduling. *ACM Transactions on Algorithms*, 6, 2009.

16 H.-L. Chan, J. Edmonds, T.-W. Lam, L.-K. Lee, A. Marchetti-Spaccamela and K. Pruhs. Nonclairvoyant speed scaling for flow and energy. *Proc. 26th International Symposium on Theoretical Aspects of Computer Science*, 255–264, 2009.

17 H.-L. Chan, J. Edmonds and K. Pruhs. Speed scaling of processes with arbitrary speedup curves on a multiprocessor. *Proc. 21st Annual ACM Symposium on Parallel Algorithms and Architectures*, 1–10, 2009.

18 E.D. Demaine, M. Ghodsi, M.T. Hajiaghayi, A.S. Sayedi-Roshkhar and M. Zadimoghaddam. Scheduling to minimize gaps and power consumption. *Proc. 19th Annual ACM Symposium on Parallel Algorithms and Architectures*, 46–54, 2007.

19 G. Greiner, T. Nonner and A. Souza. The bell is ringing in speed-scaled multiprocessor scheduling. *Proc. 21st Annual ACM Symposium on Parallel Algorithms and Architectures*, 11–18, 2009.

20 A. Gupta, R. Krishnaswamy and K. Pruhs. Scalably scheduling power-heterogeneous processors. *Proc. 37th International Colloqium on Automata, Languages and Programming*, Springer LNCS 6198, 312–323, 2010.

21 X. Han, T.W, Lam, L.-K. Lee, I.K.-K. To and P.W.H. Wong. Deadline scheduling and power management for speed bounded processors. *Theoretical Computer Science*, 411:3587-3600, 2010.

22 S. Irani, S.K. Shukla and R. Gupta. Algorithms for power savings. *ACM Transactions on Algorithms*, 3, 2007.

23 T.W. Lam, L.-K. Lee, H.-F. Ting, I.K.-K. To and P.W.H. Wong. Sleep with guilt and work faster to minimize flow plus energy. *Proc. 36th International Colloqium on Automata, Languages and Programming*, Springer LNCS 5555, 665–676, 2009.

24 T.-W. Lam, L.-K. Lee, I.K.-K. To and P.W.H. Wong. Energy efficient deadline scheduling in two processor systems. *Proc. 18th International Symposium on Algorithms and Computation*, Springer LNCS 4835, 476–487, 2007.

25 T.-W. Lam, L.-K. Lee, I.K.-K. To and P.W.H. Wong. Competitive non-migratory scheduling for flow time and energy. *Proc. 20th Annual ACM Symposium on Parallel Algorithms and Architectures*, 256–264, 2008.

26 T.-W. Lam, L.-K. Lee, I.K.-K. To and P.W.H. Wong. Speed scaling functions for flow time scheduling based on active job count. *Proc. 16th Annual European Symposium on Algorithms*, Springer LNCS 5193, 647–659, 2008.

27 M. Li, B.J. Liu and F.F. Yao. Min-energy voltage allocation for tree-structured tasks. *Journal on Combintorial Optimization*, 11:305–319, 2006.

28 M. Li and F.F. Yao. An efficient algorithm for computing optimal discrete voltage schedules. *SIAM Journal on Computing*, 35:658–671, 2005.

29 M. Li, A.C. Yao and F.F. Yao. Discrete and continuous min-energy schedules for variable voltage processors. *Proc. National Academy of Sciences USA*, 103, 3983–3987, 2006.

30 J. Markoff and S. Lohr. Intel's huge bet turns iffy. *The New York Times*, Septemer 29, 2002.

31 K. Pruhs, P. Uthaisombut and G.J. Woeginger. Getting the best response for your erg. *ACM Transactions on Algorithms*, 4, 2008.

32 K. Pruhs, R. van Stee and P. Uthaisombut. Speed scaling of tasks with precedence constraints. *Theory of Computing Systems*, 43:67–80, 2008.

33 D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

34 F.F. Yao, A.J. Demers and S. Shenker. A scheduling model for reduced CPU energy. *Proc. 36th IEEE Symposium on Foundations of Computer Science*, 374–382, 1995.

Structural Decomposition Methods and What They are Good For

Markus Aschinger[1], Conrad Drescher[1], Georg Gottlob[1,2], Peter Jeavons[1], and Evgenij Thorstensen[1]

1 Computing Laboratory, University of Oxford
2 Oxford Man Institute of Quantitative Finance
firstname.lastname@comlab.ox.ac.uk

Abstract

This paper reviews structural problem decomposition methods, such as tree and path decompositions. It is argued that these notions can be applied in two distinct ways: Either to show that a problem is efficiently solvable when a width parameter is fixed, or to prove that the unrestricted (or some width-parameter free) version of a problem is tractable by using a width-notion as a mathematical tool for directly solving the problem at hand. Examples are given for both cases. As a new showcase for the latter usage, we report some recent results on the Partner Units Problem, a form of configuration problem arising in an industrial context. We use the notion of a path decomposition to identify and solve a tractable class of instances of this problem with practical relevance.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.12

1 Introduction: Treewidth and Other Notions of Width

Tree decompositions [52, 6] and their variants and generalizations [44] constitute a significant success story of Theoretical Computer Science. In fact, tree decompositions and polynomial algorithms for bounded treewidth constitute one of the most effective weapons to attack NP hard problems, namely, by recognizing and efficiently solving large classes of tractable problem instances. Structural problem decompositions such as treewidth are thus closely related to fixed-parameter tractability [19, 37].

Tree and Path Decompositions. Formally, a *tree decomposition* of a graph $G = (V, E)$ is a pair $\mathcal{P} = (T, \chi)$ such that $T = (W, F)$ is a tree or forest, and where the function χ associates to every $w \in W$ a subset $B \subseteq V$ such that

(1) for every vertex v in V there is a vertex $w \in W$ with $v \in \chi(w)$;
(2) for every edge (v_1, v_2) in E there is a vertex $w \in W$ with $\{v_1, v_2\} \subseteq \chi(w)$; and
(3) for every vertex v in V the set $\{w \in W \mid v \in \chi(w)\}$ induces a subtree of T.

Condition (3) is called the connectedness condition. The subsets B associated with the vertices of W are called bags. The width of a tree decomposition is $\max_{w \in W}(|\chi(w)| - 1)$. The *treewidth* $tw(G)$ of G is the minimum width over all its tree decompositions.

A *path decomposition* of a graph is a tree decomposition where $T = (W, F)$ actually consists of a simple root-leaf path. The pathwidth $\mathrm{pw}(G)$ of a graph is the minimum width over all its path decompositions.

Several variants and generalizations of treewidth have been introduced, for an overview see [44]. For example, the notion of treewidth is easily generalized from graphs to finite structures. A *finite structure* $\mathcal{A}$ consists of a domain A, and relations $R_1, \ldots, R_k$ of arities $a_1, \ldots, a_k$, respectively. Each such relation R_i consists of a set of tuples $(r_1, \ldots, r_{a_i}) \in R$ where $r_1, ..., r_{a_i} \in A$. A graph $G = (V, E)$ corresponds to a finite structure whose domain is V, with a binary relation E encoding the edges. If G is undirected, then E contains both pairs (v, w) and (w, v) for each

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 12–28
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

edge $\{v, w\}$ of G. Undirected graphs are thus represented as special cases of arbitrary (possibly directed) graphs. The *Gaifman graph* of a structure $\mathcal{A}$, is the undirected graph $G(\mathcal{A})$ whose set of vertices is the domain A of $\mathcal{A}$ and where there is an edge $\{a, b\}$ iff $a, b \in A$ and $a \neq b$, and there exists a tuple in one of the relations of $\mathcal{A}$ in which a and b occur jointly. A *tree decomposition of the structure* $\mathcal{A}$ is a tree decomposition for the Gaifman graph, $G(\mathcal{A})$. The *treewidth* $tw(\mathcal{A})$ *of a structure* $\mathcal{A}$ is defined accordingly, i.e., $tw(\mathcal{A}) = tw(G(\mathcal{A}))$. Similarly, the pathwidth $pw(A)$ is defined as $pw(G(\mathcal{A})$. The treewidth $tw(C)$ (pathwidth $pw(C)$) of a class C of finite structures is the maximum over all $tw(\mathcal{A})$ ($pw(\mathcal{A})$) for $\mathcal{A} \in C$. A tree decomposition of a hypergraph H is a tree decomposition of the *primal graph* $G(H)$ of H, which has the same vertices as H and an edge between each pair of vertices that jointly occur in a hyperedge of H.

It is NP hard to determine the treewidth of a structure $\mathcal{A}$. However, for each fixed k, checking whether $tw(\mathcal{A}) \leq k$, and if so, computing a tree decomposition for $\mathcal{A}$ of optimal width, is achievable in linear time [5], and was recently shown to be achievable in logarithmic space [20]. Even though the multiplicative constant factor of Bodlander's linear algorithm [5] is exponential in k, there are algorithms that find exact tree decompositions in reasonable time or good upper approximations in many cases of practical relevance, see for example [7, 8] and the references therein.

The treewidth of a graph or relational structure is an invariant that can be used as a parameter to define infinite classes of graphs (or structures) related to problem instances. Many NP hard problems of practical relevance can be solved in polynomial time on instances of bounded treewidth and some are actually fixed-parameter tractable with respect to treewidth. Given that bounded pathwidth implies bounded treewidth, these results hold a fortiori for bounded pathwidth. The notion of treewidth is at the base of strong meta-theorems such as Courcelle's Theorem [13], which states that any problem expressible in monadic second-order logic (MSO) over structures of bounded treewidth can be solved in linear time (or, by a recent result, in logarithmic space [20]). Many problems, e.g. the 3-colorability of graphs, are very easily expressed in terms of MSO, and thus Courcelle's theorem turns out to be a very effective tool for obtaining tractability results.

Hypergraph Decompositions The structure of a computational problem is sometimes better described by a hypergraph than by a graph. Therefore, various width-notions for hypergraphs have been defined and studied, and often these are more effective than simply considering the associated primal graph. In particular, the notion of *hypertree width*, which is based on *hypertree decompositions* [24, 31, 1], is an appropriate measure for the degree of acyclicity of a hypergraph. Bounded hypertree width generalizes the concept of *α-acyclic hypergraphs* developed by Fagin [21]. In essence, a hypertree decomposition of width k for a hypergraph H can be obtained from a suitable tree decomposition of $G(H)$ by covering each bag with at most k hyperedges of H. However, under this definition, which actually defines the concept of *generalized hypertree width*, $ghw(H)$, of a hypergraph H, it is unfortunately NP-hard to determine whether a hypergraph has width $\leq k$ for $k = 3$ [32]. Therefore, to define the actual concept of hypertree width, an additional condition is imposed that ensures the tractability of computing hypertree decompositions of low width (for details, see [24]). Bounded hypertree width is strictly more general than bounded treewidth because there exist families of hypergraphs with bounded hypertree width whose treewidth is unbounded.

A number of practically relevant problems become tractable for instances whose associated hypergraphs have low hypertree width. Examples are constraint satisfaction problems (CSPs) [16, 29], see Section 3.1 for a definition, and combinatorial auctions [26]. Very roughly, CSPs of bounded hypertree-width are *loosely constrained* and therefore tractable. CSPs may also become tractable because their associated hypergraphs are –in a precise sense– *tightly constrained*. Formally this was captured by the entropy-based measure of *fractional edge cover* [39].

CSPs whose associated hypergraphs are of bounded fractional edge cover number are tightly constrained and can be solved in polynomial time. Combining hypertree decomposition with fractional edge covers yields *fractional hypertree decompositions* [39], a decomposition method that is more general than both hypertree decompositions and fractional edge covers. While hypertree decompositions will be considered again in Section 3.1, this paper mainly deals with tree and path decompositions.

Structure of the Paper. The rest of the paper is structured as follows. In the next section, we present a taxonomy of the main uses of tree decomposition in Computer Science. We will distinguish between two main categories (categories 1 and 2) and four sub-categories (1.a, 1.b, 2.a, and 2.b) of applications of tree decompositions. In Section 3, we give examples of problems that fall into the categories 1.a, 1.b, and 2.a, respectively. In section 4, we describe a relevant version of the Partner Unit Problem (PUP) and report about our recent result [3] showing that the problem falls in category 2.b and is therefore tractable.

2 Taxonomy of Main Uses of Tree Decompositions

One may distinguish between two main usages of tree decompositions, each of which can be subdivided in turn into two sub-cases. The following taxonomy will be illustrated with concrete examples in the next sections. When we speak about the treewidth of a problem instance, we mean the treewidth of some graph associated with the instance. Obviously, for each concrete problem, one has to indicate what this graph is, and, whenever necessary, how it can be obtained from the instance.

1. *Proving a problem tractable for instances of bounded treewidth.* This is probably the main use of treewidth. The idea is that many practically relevant classes of inputs actually have bounded treewidth, and that for such classes, a polynomial algorithm can be designed. We distinguish between general tractability results and fixed-parameter tractability (FPT) results:

   1.a *General tractability results.* We are able to show that the problem becomes tractable, but the best known polynomial algorithms are of complexity $\Omega(n^{f(k)})$, where $\lim_{k\to\infty} f(k) = \infty$. In many such cases it can actually be proven that the problem is *fixed-parameter intractable* with respect to the treewidth parameter k. A typical problem in this category, is the CSP problem, which we will discuss in more detail in the next section.

   1.b *Fixed-parameter tractability results.* This is the case if there exists a function f such that the problem is solvable in time $f(k) \times n^{O(1)}$ on instances of treewidth $\leq k$. If the problem is actually solvable in time $f(k) \times O(n)$, then we speak about *fixed-parameter linearity.* In particular, fixed-parameter linearity results can be obtained whenever Courcelle's Theorem can be applied. To illustrate this, we will discus the MULTICUT problem in the next section.

2. *Using treewidth as a tool for proving a problem to be generally tractable (on all instances).* There are a number of cases where treewidth is used as a tool in general tractability proofs. Again, we may distinguish between two sub-cases:

   2.a *Proving that a problem is tractable both for small and large treewidth.* For some problems having some associated implicit or explicit parameter c, it can be shown that there exists a function f such that the problem is tractable both for instances having treewidth $\leq f(c)$ and also for instances having treewidth $> f(c)$. The tractability proof for low treewidth is usually completely different from the tractability proof for high treewidth. As an example for this category, we will review the problem of checking whether a loop-free undirected graph has a cycle of length $0 \mod c$, where c is a fixed constant. This problem was shown to be tractable by Thomassen [53].

2.b *Proving that all yes-instances of a problem have bounded treewidth and that the problem is tractable for this reason.* Here, for some explicit or implicit constant parameter c associated with the problem, one is able to find a function t, such that all yes-instances of the problem must have treewidth $\leq t(c)$. Moreover, one shows that for instances of bounded treewidth the problem is polynomially solvable. Note that this is actually a very special case of Case 2.a. In fact, this means that the high-treewidth case is trivially tractable, because all instances of high treewidth are no-instances. As an example for this category, we will report in Section 4 about a new result related to the *Partner Unit Problem (*Pup*)*, a problem of industrial relevance [22].

3 Examples for Tractability Results Based on Bounded Treewidth

As announced, this section contains examples for the usages of bounded treewidth as described under categories 1.a, 1.b, and 2.a. We start with a very brief review of the Constraint Satisfaction Problem (CSP) in Section 3.1. We then describe the Multicut problem and illustrate how a nice generalization of Courcelle's Theorem can be used to show that Multicut is FPT on instances of bounded treewidth. Finally, in Section 3.3 we review Thomassen's famous result stating that it can be determined in polynomial time whether a graph has an even cycle, and more generally, whether a graph has a cycle of length $0 \mod c$ where c is a fixed constant. Our presentation of all these problems and results is necessarily very brief, but we include references to literature containing a full treatment and many more results.

3.1 The Constraint Satisfaction Problem

The efficient solution of Constraint Satisfaction Problems (CSPs) has for many years been an important goal of AI research and of related disciplines, in particular, Operations Research and Database Theory.

An instance of a *constraint satisfaction problem (CSP)* (also *constraint network*) is a triple $I = (\mathit{Var}, U, \mathcal{C})$, where Var is a finite set of variables, U is a finite domain of values, and $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$ is a finite set of constraints. Each constraint C_i is a pair (S_i, r_i), where S_i is a list of variables of length m_i called the *constraint scope*, and r_i is an m_i-ary relation over U, called the *constraint relation*. (The tuples of r_i indicate the allowed combinations of simultaneous values for the variables S_i). A *solution* to a CSP instance is a substitution $\theta : \mathit{Var} \longrightarrow U$, such that for each $1 \leq i \leq q$, $S_i\theta \in r_i$. The problem of deciding whether a CSP instance has any solution is called *constraint satisfiability (CS)*. Many well-known problems in Computer Science and Mathematics can be formulated as CSPs. For example, the famous problem of *graph three-colorability (3COL)*, i.e., deciding whether the vertices of a graph $G = \langle \mathit{Vertices}, \mathit{Edges} \rangle$ can be colored by three colors (say: red, green, blue) such that no edge links two vertices having the same color, can be formulated as a CSP as follows. The set Var contains a variable X_v for each vertex $v \in \mathit{Vertices}$. For each edge $e = \langle v, w \rangle \in \mathit{Edges}$, the set $\mathcal{C}$ contains a constraint $C_e = (S_e, r_e)$, where $S_e = \langle X_v, X_w \rangle$ and r_e is the relation $r_{\neq}$ consisting of all pairs of different colors, i.e., $r_{\neq} = \{\langle \mathit{red}, \mathit{green} \rangle, \langle \mathit{red}, \mathit{blue} \rangle, \langle \mathit{green}, \mathit{red} \rangle, \langle \mathit{green}, \mathit{blue} \rangle, \langle \mathit{blue}, \mathit{red} \rangle, \langle \mathit{blue}, \mathit{green} \rangle\}$.

It is well-known, and easy to see, that Constraint Satisfiability is an NP-complete problem. Membership in NP is obvious. NP-hardness follows immediately, e.g. from the NP hardness of 3COL. It is also well-known [4, 43, 15] that the CSP is equivalent to various database problems, e.g., to the problem of evaluating *Boolean conjunctive queries* over a relational database.

To any CSP instance $I = (Var, U, \mathcal{C})$, we associate a hypergraph $H(I) = (V, H)$, where $V = Var$, and $H = \{var(S) \mid C = (S, r) \in \mathcal{C}\}$, where $var(S)$ denotes the set of variables in the scope S of the constraint C. The graph $G(I)$ associated with a CSP is the primal graph $G(H(I))$ of the hypergraph $H(I)$. Note that if all constraints of a CSP instance I are binary, then its associated hypergraph $H(I)$ is identical to its graph $G(I)$.

The following result (for treewidth) was implicit in work of Dechter and Pearl [17] based on Freuder (see [29] for clarifications). It was explicitly stated by Kolaitis and Vardi (Theorem 5.4 in [46]) and, by Chekuri and Rajaraman[12], who considered a slightly different graph associated to a CSP-instance I. The more general version for bounded hypertree width was proven in [24].

▶ **Theorem 1.** *CSPs of bounded treewidth and CSPs of bounded hypertree width are tractable.*

Since bounded treewidth implies bounded hypertree width [24, 29], it is sufficient to consider the proof for bounded hypertree width. A detailed exposition is given in the proof of Theorem 21 of [46]. Essentially it is shown that a CSP instance I of hypertree width k can be transformed in time $O(n^k)$ into an instance I^* of size n^k, where n is the size of I, whose associated hypergraph is acyclic. I^* can then be solved by using Yannakakis' well-known method for answering acyclic queries [54]. The total time for solving I is shown to be $n^{k+1} \log n$.

It is natural to ask whether we could achieve fixed-parameter tractability (FPT) by finding a better algorithm which would allow us to get rid of the constant k in the exponent of n, and thus to replace the runtime bound by some expression $f(k) \times n^c$ for a function f and a constant c independent of k. Unfortunately, this appears to be very unlikely. In fact, the problem of evaluating Boolean conjunctive queries, which is identical to the CSP problem, is easily shown to be fixed-parameter intractable (more precisely, $W[1]$-hard) with respect to either parameter, the query size or the number of variables [50], see also [38, 18]. It is therefore a fortiori FP-intractable with respect to the treewidth parameter, given that any k-variable CSP has treewidth at most $k-1$. This makes the CSP a prime example for a problem in category 1.a of our classification.

The precise complexity of solving CSPs (or answering conjunctive queries) of bounded treewidth, or hypertree width, is as follows [30, 24]:

▶ **Theorem 2.** *Deciding satisfiability for CSP instances of bounded treewidth or bounded hypertree width is* LOGCFL*-complete.*

Given that LOGCFL is a class of highly parallelizable problems included in the well-known very low classes AC_1 and NC_2, this shows that even though FP-intractable in case of bounded treewidth, the problem is efficiently parallelizable. A concrete parallel algorithm can be obtained by combining the transformation of the original CSP instance of bounded treewidth, I, into an acyclic CSP instance I' (see [29]) (which is an AC_0-reduction) with the DB-SHUNT algorithm described in [30].

By results of Grohe, Schwentick, and Segoufin [40], for CSPs of *bounded arity* (i.e., whose constraint scopes and relations are of bounded arity, but otherwise unrestricted), it was shown that bounded treewidth is actually the best possible tractability criterion: a class of CSP instances of bounded arity is tractable if and only if it has bounded treewidth. However, when the constraints can have unbounded arity, bounded treewidth is a sub-optimal tractability criterion and is dramatically outperformed by bounded hypertree width. But, as already alluded to in the introduction, there are yet more powerful decomposition methods, such as fractional hypertree decompositions [39]. Even bounded fractional hypertree width does not seem to be an optimal structural tractability criterion. An optimal criterion was very recently established by Marx in [48], however, this criterion imposes conditions not only on the constraint scopes, but also on the constraint relations, and is thus of a different, less "structural" type. Finally, let us observe that it may be useful to consider hypertree decompositions or their generalizations, rather than just tree decompositions, even in the case of bounded arities. In fact, in Theorem 16 of [33] it was shown that each CSP instance I having n variables is of hypertree width $\leq \lfloor n/2 \rfloor + 1$, while its treewidth may actually be $n-1$.

3.2 Network Multicut Problems

We now illustrate category 1.b of our taxonomy by a number of variants of the well-known Multicut problem.

Multicut problems and their complexity. Multicut problems are highly relevant to the field of network design. The smallest size of a multicut reveals the robustness and stability of a network with multicommodity flows. Given a set $H \subseteq V^2$ of pairs of *terminal* vertices of a graph $G = (V, E)$, several forms of Multicut have been considered and have given rise to complexity studies [9, 14, 23, 42, 41, 47, 49].

A large part of the literature deals with the Edge Multicut variant in which the solution is a set of edges $E' \subseteq E$ that, if removed, separate every terminal pair. Variants in which sets of vertices are removed, such as Unrestricted Vertex Multicut, wherein any vertices can appear in the solution set V', and Restricted Vertex Multicut, wherein no terminal vertices can appear in the solution set, have also been studied [9]. Formal definitions of these versions of Multicut are given below.

In 2006, Guo et al. [41] present an algorithm which solves all three of the variants of the problem in polynomial time given *two* constant parameters, the cardinality of the set of terminal pairs, $|H|$, and the treewidth ω of G.

In [28], Gottlob and Tien Lee introduced a new single parameter for which all variants of Multicut are FPT. This unique parameter is the treewidth ω^* of the input structure $\mathcal{A} = (V, E, H)$, which is equal to the treewidth of the graph $(V, E \cup H)$. These more recent FPT-results are proved by using a powerful extended version of Courcelle's theorem due to Arnborg, Lagergren, and Seese [2]. By formulating Multicut in such an extended MSO, the considered Multicut problems are shown to be FPT with respect to ω^*. Note that if the input graph has bounded treewidth, and if H has bounded cardinality, then, obviously, the entire input structure $\mathcal{A} = (V, E, H)$ also has bounded treewidth ω^*. However, ω^* can be bounded even in cases where G has bounded treewidth but H has unbounded cardinality. The FPT results for bounded ω^* are thus a strict generalization of the results by Guo et al. [41]. These findings demonstrate that powerful logical tools such as the extended version of Courcelle's master theorem by Arnborg, Lagergren, and Seese [2] can be applied to advance the state of the art in network multicut theory and can help identify parameters of interest in complexity analysis.

Formal definitions of multicut problems. We now define the various different versions of the MULTICUT problem mentioned earlier. The Edge Multicut problem is formally defined as follows:

▶ **Definition 3.** Edge Multicut (EMC)

Input: An undirected graph $G = (V, E)$, and $H \subseteq V \times V$ a collection of pairs of vertices.
Task: Find a minimal cardinality set $E' \subseteq E$ whose removal disconnects each pair in H.

The vertex variants of the problem were identified by Calinescu et al. [9] and they are defined as:

▶ **Definition 4.** Unrestricted Vertex Multicut (UVMC)

Input: An undirected graph $G = (V, E)$, and $H \subseteq V \times V$ a collection of pairs of vertices.
Task: Find a minimal cardinality set $V' \subseteq V$ whose removal disconnects each pair in H.

If H is a set of pairs of elements, we denote by H_0 the set of all elements occurring in some pair of H, i.e., $H_0 = \{x | \exists y : (x, y) \in H \vee (y, x) \in H\}$.

▶ **Definition 5.** Restricted Vertex Multicut (RVMC)

Input: An undirected graph $G = (V, E)$, and $H \subseteq V \times V$, a collection of pairs of vertices.
Task: Find a minimal cardinality subset $V' \subseteq V$ where $V' \cap H_0 = \emptyset$ and whose removal disconnects each pair of vertices in H.

It has been shown that all three forms of the problem are NP-complete for general graphs [14, 9] and remain hard even for graphs with bounded treewidth [23, 9].

A strong master theorem by Arnborg, Lagergren, and Seese. Before showing that the problems become FPT for inputs of bounded treewidth ω^*, we present a strong and most useful generalization of Courcelle's Theorem by Arnborg, Lagergren, and Seese [2]. Let us refer to the version of MSO where second-order quantification is restricted to *sets of domain elements* (e.g., sets of vertices of the input graph) as MSO_1. Note that Courcelle [13] and other authors considered an extended version of MSO called MSO_2 which extends MSO_1 by the possibility of quantifying over subsets of any relation of the input structure, e.g., sets of edges of an input graph. Thus, for example, if a relational symbol R is part of the problem signature, then a subformula $(\exists X \subseteq R)\varphi(X)$, expressing that there exists a subset X of the relation R such that $\varphi(X)$ holds for some formula φ, could be part of an MSO_2 formula. Courcelle's Theorem is actually valid for MSO_2 (and thus, in particular, for MSO_1). Arnborg, Lagergren and Seese [2] considered an important extension of MSO_2, called *extended MSO* that can be used to formulate optimization and counting problems. They proved that solving problems expressible in this form over input structures is FPT with respect to the treewidth of these input structures.

While in the original setting in [2], extended MSO properties were defined in a much more general context, it is sufficient for our purposes to state a drastically simplified definition and, accordingly, a simplified master theorem (Theorem 7). By *optimization* we here understand the search for a minimum or maximum solution according to some cardinality criteria. The solution itself is an "optimal" assignment of sets to second-order variables that freely occur in some MSO formula, such that the formula is satisfied over a given input structure. More precisely:

▶ **Definition 6** (simplified version of a definition in [2]). An optimization problem is an *extended MSO cardinality optimization problem* if it can be expressed in the following form. The input of the problem is a a structure $\mathcal{A} = (V, E, H)$, where V is a set (the universe of $\mathcal{A}$), and E and H are binary relations over elements of V. Let $\varphi(X)$ be a fixed MSO_1 or MSO_2 formula over $\mathcal{A}$, where X is either a free set variable ranging over subsets of V, or a binary relation variable ranging over subsets of E. The task is to find an assignment[1] among all possible assignments z' to the variable X such that:

$$|z(X)| \;=\; opt\{|z'(X)| : (\mathcal{A}, z') \models \varphi(X)\}$$

where *opt* is either min or max. A suitable assignment z is called a *solution* to the extended MSO cardinality optimization problem.

Using this definition, Arnborg, Lagergren and Seese found the following important fixed-parameter tractability result [2]:

▶ **Theorem 7.** *Solving extended MSO cardinality problems is fixed-parameter tractable w.r.t. the treewidth of the input structure. In particular, for a fixed extended MSO cardinality optimization problem P, and a class C of input structures whose treewidth is bounded by some constant, the following can be done in linear time:*

- *Determine whether P has a solution for an input from C.*
- *Compute a solution for an input from C, if one exists.*

Applying the master theorem to multicut problems. We first define a useful formula that states that two vertices x and y are connected by a path that lies entirely in a set S of vertices.

[1] An assignment z to the variable X is an interpretation of X that maps X to a subset $z(X)$ of V if X is unary and a subset of E if X is binary.

▶ **Definition 8.** On structures $\mathcal{A} = (V, E, H)$ as above, let *connects*(S, x, y) be defined as follows:

$$S(x) \wedge S(y) \wedge \forall P\Big(\big(P(x) \wedge \neg P(y)\big) \rightarrow \big(\exists v \exists w\, (S(v) \wedge S(w) \wedge P(v) \wedge \neg P(w) \wedge E(v, w))\big)\Big).$$

▶ **Lemma 9** ([28]). *Over a structure $\mathcal{A} = (V, E, H)$ as above, the formula connects(S, x, y) states that there is a path in (V, E) that connects vertex x to vertex y, and this path lies entirely in S. In particular, this is also true for directed graphs.*

▶ **Theorem 10** ([28]). *The problems* UVMC, RVMC, *and* EMC *are fixed parameter linear with respect to the treewidth ω^* of the input structure (V, E, H).*

Proof. (Sketch.) We outline the proof for UVMC. The problem UVMC can in fact be expressed as an extended MSO cardinality optimization problem in the following way: Find an assignment $z \subseteq V$ to set variable X such that $|z(X)| = min\{|z'(X)| : \langle V, E, H, z'\rangle \models uvmc(X)\}$, where $uvmc(X)$ is an MSO_1 expression defined as:

$$uvmc(X) \quad \equiv \quad \forall x \forall y \Big(H(x, y) \rightarrow \forall S\big(\text{connects}(S, x, y) \rightarrow \exists v(X(v) \wedge S(v))\big)\Big).$$

In words, the formula $uvmc(X)$ defines X to be such that for each pair $(x, y) \in H$ (i.e., for each pair (x, y) that must be disconnected), whenever there is a set S of vertices from V that contains a path from x to y, then X must intersect S, i.e. contain some vertex from S. In [28] it is formally proven that $uvmc(X)$ is true iff the set X is an unrestricted vertex multicut of (V, E, H). The theorem then follows immediately from Theorem 7. Slight variants of this proof yield the corresponding FPT results for RVMC and EMC. For *EMC*, quantification over subsets of the edge relation is used. For details, see [28]. ◀

Master theorems such as Courcelle's and the one of Arnborg, Lagergren and Seese are constructive and can be used for the implementation of model-checking tools such as MONA [45] that directly interpret an MSO-formulation of a problem, or directly compile an MSO-formula into a solution algorithm. However, in order to obtain more efficient algorithms and better upper bounds it is currently still advisable to attempt a more detailed *ad hoc* analysis of the problem at hand, once the master theorems show us they are FPT. The above FPT-results, for example, were used by Pichler, Rümmele and Woltran [51] as a starting point for the design of very effective algorithms and for the derivation of rather low complexity bounds for MULTICUT problems on inputs of low treewidth. In the future we may expect new tools that are able to automatically compile algorithms and bounds of a similar quality. As an intermediate step, we believe that it is rewarding to replace MSO by equally expressive but much simpler (and better optimizable) languages for expressing a problem. One such candidate is the monadic fragment of the well-known *Datalog* language [11, 10]. It was recently shown in [35] that over structures of bounded treewidth, when a tree decomposition is provided, monadic Datalog is exactly as expressive as MSO. As illustrated in [34, 36], many problems of practical relevance can be easily encoded in monadic Datalog, and special interpreters that execute monadic Datalog programs over structures of bounded treewidth turned out to be much more efficient than MONA fed with the MSO formulas that are logically equivalent to those Datalog programs. For a further discussion, see Section 5 of [34].

3.3 The Even Cycle Problem

The EVEN CYCLE PROBLEM (ECP) is the problem of determining whether a loop-free undirected graph has an even cycle. The tractability of ECP was open for a long time, until Carsten Thomassen proved it polynomial [53]. Actually, Thomassen proved the following more general result regarding the problems mCP of whether a loop-free undirected graph has a cycle of length 0 modulo m, where m is a fixed positive integer.

▶ **Theorem 11** ([53]). *For each integer $m > 0$, mCP is decidable in polynomial time.*

As the following proof outline shows, the proof is according to the pattern of Case 2.a in our taxonomy.

Proof. (Outline.) First it is shown that on graphs $G = (V, E)$ of bounded treewidth, mCP is tractable. With Courcelle's theorem this is very easy; it is sufficient to note that mCP can be expressed in MSO. (Given that Courcelle's Theorem was not known, Thomassen gave a slightly more involved *ad hoc* proof.)

The second part deals with input graphs of "large" treewidth. It is proven that for each fixed m, a number $t(m)$ can be determined, such that all graphs of treewidth $> t(m)$ must actually have a cycle whose length is a multiple of m. Thus, for the specific problem mCP, all instances of "high" treewidth are actually yes-instances. In particular, it is shown by using Robertson's and Seymour's result [52] that $t(m)$ can always be chosen large enough such that G must contain a subdivision of a grid H, which, in turn, must contain a cycle whose length is a multiple of m. ◀

In a similar fashion, while classifying the complexity of model checking for all prefix-classes of existential second-order logic (ESO) over graphs, it was shown in [27] that evaluating fixed closed formulas of type $\exists R_1, \ldots\ R_k \forall x \exists y\, \phi(V, E, R_1, \ldots, R_k, x, y)$ over a loop-free undirected input graph (V, E), where $R_1 \ldots, R_k$ are existentially quantified relation symbols of arbitrary arity, and where $\phi(V, E, R_1, \ldots, R_k, x, y)$ is a quantifier-free first order formula, is feasible in polynomial time. Note that this generalizes Thomassen's theorem, given that for each m, the problem mCP can be expressed by an ESO formula of this type.

4 The Partner Units Problem

In this section, we describe the Partner Units Problem (Pup). First, in Section 4.1, a general version of the problem is given, which is, however, intractable. In Section 4.2 we describe a special version of the Pup which is of particular industrial relevance. It is for this special case that we were able to establish tractability by exploiting the result that all yes-instances must necessarily have bounded pathwidth (and thus bounded treewidth). It is thus the special case which serves as a paradigmatic example of a problem in category 2.b of our taxonomy. In Section 4.2.4 we then report on a prototypical implementation for the special case that already could solve benchmark instances beyond the reach of the heuristic methods and "engineering approaches" previously used to solve this problem. This section gives only a short summary; the original work underlying this section, as well as detailed proofs can be found in [3].

4.1 Definition of the Partner Units Problem and Basic Facts

The Partner Units Problem (Pup) has recently been proposed as a new benchmark configuration problem [22]. It captures the essence of a specific type of configuration problem that frequently occurs in industry.

Informally it can be described as follows: Consider a set of sensors that are grouped into zones. A zone may contain many sensors, and a sensor may be attached to more than one zone. The Pup consists of connecting the sensors and zones to control units. These control units can be connected to the same fixed maximum number *UnitCap* of zones and sensors.[2] Moreover, if a sensor is attached to a zone, but the sensor and the zone are assigned to different control units,

[2] For ease of presentation we assume that *UnitCap* is the same for zones and sensors.

then the two control units in question have to be (directly) connected. However, a control unit cannot be connected to more than *InterUnitCap* other control units (the partner units).

The PUP occurs e.g. in the following application domain: Consider a museum where we want to keep track of the number of visitors that populate certain parts (zones) of the museum. To this end the doors leading from one zone to another are equipped with sensors. To keep track of the visitors the zones and sensors are attached to control units; the adjacency constraints on the control units ensure that communication between control units can be kept simple. It is worth pointing out that the PUP is not limited to this application domain: It occurs whenever sensors that are grouped into zones have to be attached to control units, and communication between units should be kept simple.

Figure 1 shows a PUP instance and a solution for the case $\mathit{UnitCap} = \mathit{InterUnitCap} = 2$ — six sensors (left) and six zones (right) which are completely inter-connected are partitioned into units (shown as squares) respecting the adjacency constraints. Note that for the given parameters this is a maximal solvable instance; it is not possible to connect a new zone or sensor to any of the existing ones.

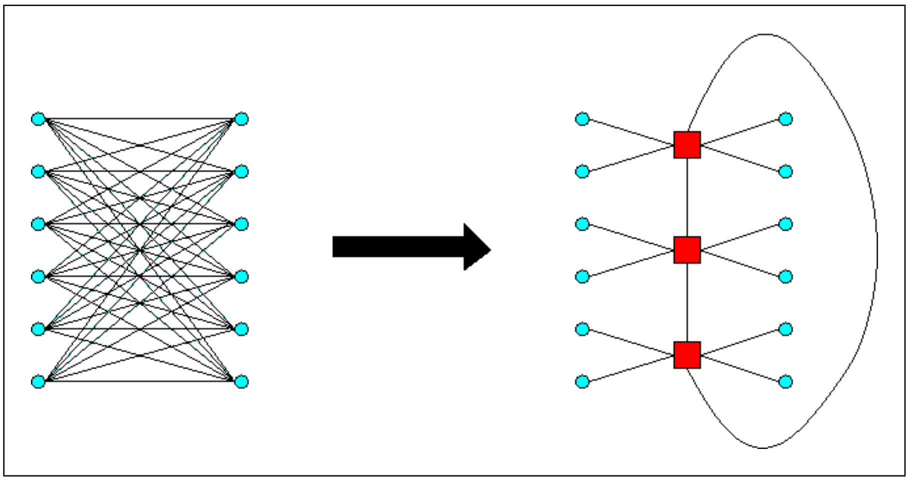


Figure 1 Solving a $K_{6,6}$ Partner Units Instance — Partitioning Sensors and Zones into Units

More formally, the PUP consists of partitioning the vertices of a bipartite graph $G = (V_1, V_2, E)$ into a set U of bags such that each bag

- contains at most *UnitCap* vertices from V_1 and at most *UnitCap* vertices from V_2; and
- has at most *InterUnitCap* adjacent bags where the bags U_1 and U_2 are adjacent whenever $v_i \in U_1$ and $v_j \in U_2$ and $(v_i, v_j) \in E$.

To every solution of the PUP we can associate a solution graph. For this we associate to every bag $u \in U$ a vertex $u' \in U'$. Then the solution graph G^* has the vertex set $V_1 \cup V_2 \cup U'$ and the set of edges $\{(v, u') \mid v \in u \wedge u \in U\} \cup \{(u'_i, u'_j) \mid u_i \text{ and } u_j \text{ are adjacent.}\}$. In the following we will refer to the subgraph of the solution graph induced by the u' as the *unit graph*.

The reasoning tasks for PUP instances that we consider in this paper are the following:

- Decide whether there is a solution (PUDP).
- Find a solution (PUSP).
- Find an optimal solution; i.e. one that uses the minimal number of control units (PUOP).

The rationale behind the optimization version is that (a) units are expensive, and (b) connections are cheap. Especially the case where the maximum number *InterUnitCap* of connections between units is limited to two is of great interest for our partners in industry.

By a reduction from BinPacking it can be shown that Puop is NP-complete [3] when *InterUnitCap* $= 0$, and *UnitCap* is part of the input. We also observe [3] that a Pup instance has no solution if it contains $K_{1,n}$ or $K_{n,1}$ as a subgraph, where $n = ((\mathit{InterUnitCap}+1)*\mathit{UnitCap})+1$.

4.2 A Special Case: $\mathit{InterUnitCap} = 2$

We now turn to the announced special case, which consists of Pups where *InterUnitCap* $= 2$, i.e. the number of neighbors of any given unit in a solution is bounded by 2. This special version of the Pup, which is of high industrial relevance, can be directly tackled. We will do this by giving an algorithm that decides this version of the Pup in NLogSpace by exploiting the notion of a path decomposition of a given graph.

For ease of presentation in the sequel we make the simplifying assumption that the underlying bipartite graph is connected. This does not affect solutions of the Pudp and the Pusp, where the connected components can be tackled independently. For optimal solutions, however, the connected components of an underlying graph will have to be considered simultaneously; cf. the discussion in section 4.2.3.

4.2.1 Basic Properties of the Pup (Special Case)

We proceed by identifying basic properties of the Pup in the special case. The key observation is that the units and their interconnections form a special kind of unit graph in any solution: either a simple path, or a simple cycle. This holds because each unit is connected to at most two partner units. Moreover, cycles are more general unit graphs than paths: Every solution can be extended to a cyclic solution; hence in the sequel we only consider cyclic solutions.

Exploiting this observation we can transform the Pup into the problem of finding a suitable path decomposition $\mathcal{P}$ of the zones-and-sensors-graph G:

▶ **Theorem 12** (Pup is Path-Decomposable). *Assume a* Pup *instance given by a graph* $G = (V_1, V_2, E)$ *is solvable with a solution graph* G^* *with* $|U| = n$*. Let* f *be the unit function that associates vertices from* G *to* U*. Then there is a path decomposition* $\mathcal{P} = (P, \chi)$ *of* G *of pathwidth* $\leq (3 * 2 * \mathit{UnitCap}) - 1$*, with the following special properties:*

(a) *The length of* P *is* $n-1$*;* $P = w_1, \dots w_{n-1}$*.*
(b) *There are sets* $S_1 \subseteq V_1$*,* $S_2 \subseteq V_2$ *with* $|S_i| \leq \mathit{UnitCap}$ *such that* $S_1 \cup S_2$ *are in every bag of* $\mathcal{P}$*.*
(c) *Apart from* $S_1 \cup S_2$ *each bag contains at most* $2 * \mathit{UnitCap}$ *elements from* V_1 *(or* V_2*, respectively).*
(d) *For any vertex* $v \in V_1 \cup V_2$ *all neighbors of* v *appear in three consecutive bags of* $\mathcal{P}$ *(assuming the first and last bag to be connected).*
(e) *For each bag* $\chi(w_i)$ *of* $\mathcal{P}$ *it holds* $\chi(w_i) = f^{-1}(U_1) \cup f^{-1}(U_i) \cup f^{-1}(U_{i+1})$ *for* $1 \leq i \leq n-1$*.*
(f) $S_1 = f^{-1}(U_1) \cap V_1$ *and* $S_2 = f^{-1}(U_1) \cap V_2$*.*

Proof. If G is solvable then there is a solution G^* whose unit graph is a cycle $U_1, \dots, U_n, U_1$. Consider $\mathcal{P} = (P = w_1, \dots, w_{n-1}, \chi)$ where $\chi(w_i) = f^{-1}(U_1) \cup f^{-1}(U_i) \cup f^{-1}(U_{i+1})$. This $\mathcal{P}$ is indeed a path decomposition:

- Every edge (v_1, v_2) is in some bag. Assume v_1 and v_2 are assigned to two different connected units U_i and U_{i+1}. Then $\{v_1, v_2\} \subseteq \chi(w_i)$.
- The connectedness condition is satisfied: For the vertices connected to unit U_1 the induced subgraph is P. All other vertices occur in at most two consecutive bags.
- Every bag in $\mathcal{P}$ contains $\leq (3 * 2 * \mathit{UnitCap})$ elements; hence $\mathrm{pw}(\mathcal{P}) \leq (3 * 2 * \mathit{UnitCap}) - 1$.

An optimal path decomposition of the complete bipartite graph $K_{n,n}$ with $n = 3 * \mathit{UnitCap}$ has width $(3 * 2 * \mathit{UnitCap}) - 1$; cf. figure 1. Hence the bound is tight. The conditions $(a - f)$ are easily seen to hold for the path decomposition $\mathcal{P}$ constructed above. ◀

Intuitively, the vertices in the sets S_1 and S_2 from condition (b) above are those that close the cycle (i.e. that are connected to unit U_1). These have to be in every bag as some of their neighbors might only appear on the last unit U_n.

4.2.2 An Algorithm for the Special Case

By Theorem 12 we know that if a Pup instance is solvable then there is a path decomposition with specific properties. But we still need an algorithm for finding such suitable path decompositions. Many algorithms for finding path decompositions of bounded width have been proposed in the literature. But for the Pup we want to find path decompositions $\mathcal{P}$ with specific properties:

- The paths should be short (the number of bags reflects the number of units); and hence,
- The bags should be rather full (in "good" solutions the units will be filled up).
- The construction of the bags must be interleaved with checking the additional constraints.

Below we introduce a novel algorithm that fits the bill; it is inspired by the algorithm for finding hypertree decompositions from [24]. This non-deterministic algorithm does the following: The bags on the path decomposition are guessed. The initial bag partitions the graph into a set of remaining components that are recursively processed simultaneously. A single bag suffices to remember which part of the graph has already been processed; the bag *separates* the processed part of the graph from the remaining components. Consequently, the current bag and the remaining components can be stored in in logarithmic space, and the algorithm runs in NLogSpace. In addition to the bags the unit function is guessed, too. According to condition $(2d)$ of Theorem 12 all neighbors of any vertex in G occur in three consecutive bags in $\mathcal{P}$. Hence, for checking locally that the unit function is correct it suffices to remember three bags at each step.

A closer look reveals that it actually is enough to remember only U_1 and two "first" and "second" units U_{i-1} and U_i. At each step the current bag's content is then given by the union of U_1 with $U_{i-1} \cup U_i$. For the next step a third unit U_{i+1} is guessed. All neighbors of vertices assigned to the current second unit U_i are guaranteed to appear in $U_{i-1} \cup U_i \cup U_{i+1}$. For the current first unit U_{i-1} this will already have been established (if $i > 2$); hence, in the next step the new current first and second unit U_i and U_{i+1} together with U_1 are again a proper separator. The neighbors of U_1, however, are only guaranteed to appear somewhere on the first, second, or last unit. Upon termination the current "second" unit is the last unit in the cycle. But in addition to the first unit U_1 the second unit U_2 has to be stored throughout a run of the algorithm, too:

```
DecidePup($G$)
Guess disjoint non-empty $U_1, U_2 \subseteq V(G)$ with $|U_i \cap V_1| \leq UnitCap \geq |U_i \cap V_2|$
$C_R \leftarrow G \setminus (U_1 \cup U_2)$
if DecidePup $(C_R, \langle U_1, U_2 \rangle, \langle U_1, U_2 \rangle)$
    then ACCEPT
    else REJECT
```

```
DecidePup($C_R, \langle U_1, U_2\rangle, \langle U_{i-1}, U_i\rangle$)
if $C_R = \emptyset$
    then
        if $\forall v \in U_1\ \mathrm{nb}(v) \subseteq U_1 \cup U_2 \cup U_i$ and
        $\forall v \in U_i\ \mathrm{nb}(v) \subseteq U_{i-1} \cup U_i \cup U_1$
            then ACCEPT
            else REJECT
    else
        Guess non-empty $U_{i+1} \subseteq V(\bigcup C_R)$ with $|U_{i+1} \cap V_1| \leq \mathit{UnitCap} \geq |U_{i+1} \cap V_2|$
        For $v \in U_i$ check $\mathrm{nb}(v) \subseteq (U_{i-1} \cup U_i \cup U_{i+1})$
        $C'_R \leftarrow (C_R \setminus U_{i+1})$
        DecidePup ($C'_R, \langle U_1, U_2\rangle, \langle U_i, U_{i+1}\rangle$)
```

Using this algorithm in [3] we show the following:

▶ **Theorem 13** (Tractability of PUDP). *The decision problem for the* PUP *is solvable by the algorithm* DECIDEPUP *in* NLOGSPACE *for InterUnitCap* $= 2$ *and any given fixed value of UnitCap.*

Answer Extraction

For actually obtaining a solution to a PUP instance we face the following problem: In general it is not possible to remember the contents of all the bags in logarithmic space. Theoretically this problem can be solved as follows: On a first accepting run of DECIDEPUP we clearly can remember the first bag's contents in logarithmic space. We can then run DECIDEPUP again with a fixed first bag, and so forth. Hence the following holds:

▶ **Theorem 14** (Tractability of PUSP). *The problem of finding a solution to the* PUP *is solvable in* NLOGSPACE *for InterUnitCap* $= 2$ *and any given fixed value of UnitCap.*

Note that the problem of answer extraction disappears when actually implementing the non-deterministic algorithm on a deterministic computer; cf. section 4.2.4.

Towards an Efficient Algorithm

We next make a number of observations that can be exploited to turn DECIDEPUP into a practically efficient algorithm.

Guiding the Guessing Not all zones and sensors assigned to units have to be chosen randomly. At most *UnitCap* neighbors of sensors and zones on the first unit can be assigned to the last unit. Hence the following holds:[3]

$$|\mathrm{nb}_s(U_1) \setminus (U_1 \cup U_2)| \leq \mathit{UnitCap} \geq |\mathrm{nb}_z(U_1) \setminus (U_1 \cup U_2)|.$$

Moreover, the neighbors of U_1 not assigned to U_1 or U_2 may only be guessed in the last step, where the number of unprocessed sensors (or zones) is at most *UnitCap*.

Starting from $i \geq 2$ we have the stronger:

$$(\mathrm{nb}_s(U_i) \setminus (U_i \cup U_{i-1})) \subseteq U_{i+1} \supseteq (\mathrm{nb}_z(U_i) \setminus (U_i \cup U_{i-1})).$$

Finding Optimal Solutions First Next recall that "good" solutions correspond to short path decompositions with filled-up bags. Moreover, the number of units used in the solution of a

[3] We denote by $\mathrm{nb}_s(U)$ the set of sensors adjacent to vertices in U.

PUP instance $G = (V_1, V_2, E)$ can always be bounded by $lb = \lceil \frac{\max(|V_1|,|V_2|)}{UnitCap} \rceil$ from below and $ub = \max(|V_1|, |V_2|)$ from above [3]. Hence we can apply iterative deepening search: First, try to find a solution with lb units; if that fails increase lb by one. This has the effect that the first solution found will be optimal. This yields the following:

▶ **Corollary 15** (Tractability of PUOP). *On connected input graphs the optimization problem for the* PUP *is solvable in* NLOGSPACE.

In this context let us point out that branch-and-bound-search (on the number of units used) does not work: E.g. a $K_{6,6}$ graph does not admit solutions with more than three units.

Symmetry Breaking We already observed that cycles are more general unit graphs than paths. But with cycles for unit graphs there is rotational symmetry: For a solution with unit graph $U_1, \ldots, U_n, U_1$ there is a solution $U_2, \ldots, U_n, U_1, U_2$, etc.. We can break this symmetry without additional computational cost by requiring that

- the first sensor is assigned to unit U_1; and
- the second sensor appears somewhere on the first half of the cycle.

4.2.3 PUOP and Multiple Connected Components

Next let us discuss the problem of finding optimal solutions when the input graph consists of more than one connected component. Here, part of the problem is that any two connected components may either have to be assigned to the same, or to two distinct unit graph(s). A priori it is unclear which of the two choices leads to better results. E.g. if we assume that $UnitCap = 2$ then two $K_{3,3}$ should be placed on one cyclic unit graph, while two $K_{6,6}$ must stand alone. In [3] we are able to show the following:

▶ **Theorem 16** (Tractability of PUOP on Multiple Connected Components). *For $InterUnitCap = 2$ and any given value of $UnitCap$ the optimization problem for the* PUP *on multiple connected components is solvable in* NLOGSPACE *if there are only logarithmically many connected components in the input graph.*

4.2.4 Implementation and Evaluation

We prototypically implemented the DECIDEPUP algorithm in Java, replacing the non-determinism by a backtracking search mechanism. A detailed description of the procedure is beyond the scope of this paper. However, in [25] a deterministic backtracking version of the non-deterministic hypertree decomposition algorithm from [24] is described, and the issues we face when making DECIDEPUP deterministic are very similar. Suffice it to say the following: To avoid repeated sub-computations we store those pairs of bags and remaining components (represented by unassigned neighbors) that could not be decomposed. We don't store successful pairs — the first such pair occurs when finding a solution. As there are only polynomially many such pairs the overall runtime of the algorithm is polynomial [3]. Finally observe that for the backtracking search we have to store the choices made, and hence answer extraction is easy.

We have evaluated our algorithm on a set of benchmark instances that we received from our partners in industry. Using our prototypical implementation we could solve many instances that were beyond the reach of the previously used heuristic methods and engineering approaches [3].

5 Conclusion

In this work we have reviewed structural problem decomposition methods, such as path-, tree-, and hypertree decompositions. We have introduced a taxonomy of usages of treewidth for proving tractability results, and illustrated each category by an example. In particular, we have shown that treewidth (or pathwidth) can be applied in two distinct main ways: Either to show that a problem is efficiently solvable when a width parameter is fixed, or to prove that the unrestricted (or some width-parameter free) version of a problem is tractable by using a width-notion as a mathematical tool for directly solving the problem at hand.

As a show case for the latter usage we have reported on some recent results concerning the Partner Units Problem, a type of configuration problem that was proposed to us by an industrial partner. We have shown that, while the Pup is intractable in general, the notion of a path decomposition can be used to obtain a polynomial algorithm for a highly relevant special case. Our prototypical implementation of the respective DecidePup algorithm could solve many previously unsolvable problem instances.

There is still significant work to be done on the Pup:

- We need to analyze the cases with $\mathit{InterUnitCap} = k$ and $\mathit{UnitCap} = m$ for fixed constants $k > 2, m$.
- We would like to find better algorithms for the NP-hard general case (when *InterUnitCap* and *UnitCap* are unbounded).
- We have not yet exploited heuristics for the search.

Acknowledgment. Work funded by EPSRC Grant EP/G055114/1 "Constraint Satisfaction for Configuration: Logical Fundamentals, Algorithms and Complexity. G. Gottlob's work, in particular, with respect to specific topics arising in the context of the co-operation with Siemens Austria and the University of Klagenfurt, was also partially funded by the FFG Fit-IT project Reconcile. G. Gottlob would also like to acknowledge the Royal Society Wolfson Research Merit Award.

References

1 Isolde Adler, Georg Gottlob, and Martin Grohe. Hypertree width and related hypergraph invariants. *Eur. J. Comb.*, 28(8):2167–2181, 2007.

2 Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

3 M. Aschinger, C. Drescher, G. Gottlob, P. Jeavons, and E. Thorstensen. Tackling the Partner Units Problem. Technical Report RR-10-28, Computing Laboratory, University of Oxford, 2010. Available from the authors.

4 Wolfgang Bibel. Constraint satisfaction from a deductive viewpoint. *Artif. Intell.*, 35(3):401–413, 1988.

5 Hans L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, STOC '93, pages 226–234, New York, NY, USA, 1993. ACM.

6 Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

7 Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. On exact algorithms for treewidth. In *Algorithms - ESA 2006, 14th Annual European Symposium, Zurich, Switzerland, September 11-13, 2006, Proceedings*, pages 672–683, 2006.

8 Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations i. upper bounds. *Inf. Comput.*, 208(3):259–275, 2010.

9 Gruia Calinescu, Cristina G. Fernandes, and Bruce A. Reed. Multicuts in unweighted graphs with bounded degree and bounded tree-width. *J. Algorithms*, 48(2):333–359, 2003.

10 Stefano Ceri, Georg Gottlob, and Letizia Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Trans. Knowl. Data Eng.*, 1(1):146–166, 1989.

11 Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases.* Springer, 1990.

12 Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2):211–229, 2000.

13 Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

14 Elias Dahlhaus, David S. Johnson, Christos H. Papadimitriou, Paul D. Seymour, and Mihalis Yannakakis. The complexity of multiterminal cuts. *SIAM J. Comput.*, 23(4):864–894, 1994.

15 Rina Dechter. From local to global consistency. *Artif. Intell.*, 55(1):87–108, 1992.

16 Rina Dechter. *Constraint Processing.* Morgan Kaufmann, 2003.

17 Rina Dechter and Judea Pearl. Tree-clustering schemes for constraint-processing. In *AAAI*, pages 150–154, 1988.

18 R.G. Downey, M.R. Fellows, and U. Taylor. The parameterized complexity of relational database queries and an improved characterization of W[1]. *Complexity, and Logic, volume 39 of Proceedings of DMTCSJ*, pages 149–213, 1996.

19 Rodney G. Downey and Michael R. Fellows. *Parameterized Complexity.* Springer, 1999.

20 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 143–152, 2010.

21 Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983.

22 A. Falkner, A. Haselböck, and G. Schenner. Modeling Technical Product Configuration Problems. In *Proceedings of the Workshop on Configuration at ECAI 2010*, pages 40 – 45, Lisbon, Portugal, 2010.

23 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Primal-dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 181:3–20, 1997.

24 G. Gottlob, N. Leone, and F. Scarcello. Hypertree Decomposition and Tractable Queries. *Journal of Computer and System Sciences*, 64(3):579–627, 2002.

25 G. Gottlob and M. Samer. A backtracking-based algorithm for hypertree decomposition. *ACM Journal of Experimental Algorithmics*, 13, 2008.

26 Georg Gottlob and Gianluigi Greco. On the complexity of combinatorial auctions: structured item graphs and hypertree decomposition. In *ACM Conference on Electronic Commerce*, pages 152–161, 2007.

27 Georg Gottlob, Phokion G. Kolaitis, and Thomas Schwentick. Existential second-order logic over graphs: Charting the tractability frontier. *J. ACM*, 51(2):312–362, 2004.

28 Georg Gottlob and Stephanie Tien Lee. A logical approach to multicut problems. *Inf. Process. Lett.*, 103(4):136–141, 2007.

29 Georg Gottlob, Nicola Leone, and Francesco Scarcello. A comparison of structural CSP decomposition methods. *Artif. Intell.*, 124(2):243–282, 2000.

30 Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *J. ACM*, 48(3):431–498, 2001.

31 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions: A survey. In *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001 Marianske Lazne, Czech Republic, August 27-31, 2001, Proceedings*, pages 37–57, 2001.

32 Georg Gottlob, Zoltán Miklós, and Thomas Schwentick. Generalized hypertree decompositions: NP-hardness and tractable variants. *J. ACM*, 56(6), 2009.

33 Georg Gottlob and Alan Nash. Efficient core computation in data exchange. *J. ACM*, 55(2), 2008.

34 Georg Gottlob, Reinhard Pichler, and Fang Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artif. Intell.*, 174(1):105–132, 2010.

35 Georg Gottlob, Reinhard Pichler, and Fang Wei. Monadic datalog over finite structures of bounded treewidth. *ACM Trans. Comput. Log.*, 12(1):3, 2010.

36 Georg Gottlob, Reinhard Pichler, and Fang Wei. Tractable database design and datalog abduction through bounded treewidth. *Inf. Syst.*, 35(3):278–298, 2010.

37 Martin Grohe. Descriptive and parameterized complexity. In J. Flum and M. Rodriguez-Artalejo, editors, *Computer Science Logic CSL'99*, volume 1683, pages 14–31. Springer, 1999.

38 Martin Grohe. The parameterized complexity of database queries. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, pages 82–92, New York, NY, USA, 2001. ACM.

39 Martin Grohe and Dániel Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298, 2006.

40 Martin Grohe, Thomas Schwentick, and Luc Segoufin. When is the evaluation of conjunctive queries tractable? In *STOC*, pages 657–666, 2001.

41 Jiong Guo, Falk Hüffner, Erhan Kenar, Rolf Niedermeier, and Johannes Uhlmann. Complexity and exact algorithms for multicut. In Jirí Wiedermann, Gerard Tel, Jaroslav Pokorný, Mária Bieliková, and Julius Stuller, editors, *SOFSEM*, volume 3831 of *Lecture Notes in Computer Science*, pages 303–312. Springer, 2006.

42 Jiong Guo and Rolf Niedermeier. Fixed-parameter tractability and data reduction for multicut in trees. *Netw.*, 46(3):124–135, 2005.

43 Marc Gyssens, Peter Jeavons, and David A. Cohen. Decomposing constraint satisfaction problems using database techniques. *Artif. Intell.*, 66(1):57–89, 1994.

44 Petr Hlinený, Sang il Oum, Detlef Seese, and Georg Gottlob. Width parameters beyond tree-width and their applications. *Comput. J.*, 51(3):326–362, 2008.

45 Nils Klarlund, Anders Møller, and Michael I. Schwartzbach. Mona implementation secrets. *Int. J. Found. Comput. Sci.*, 13(4):571–586, 2002.

46 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci.*, 61(2):302–332, 2000.

47 Dániel Marx. Parameterized graph separation algorithms. *Theoretical Computer Science*, 351(3):394–406, 2006.

48 Dániel Marx. Tractable hypergraph properties for constraint satisfaction and conjunctive queries. In *Proceedings of the 42nd ACM Symposium on Theory of Computing*, pages 735–744, 2010.

49 Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. *CoRR*, abs/1010.3633, 2010.

50 Christos H. Papadimitriou and Mihalis Yannakakis. On the complexity of database queries. *J. Comput. Syst. Sci.*, 58(3):407–427, 1999.

51 Reinhard Pichler, Stefan Rümmele, and Stefan Woltran. Multicut algorithms via tree decompositions. In *Algorithms and Complexity, 7th International Conference, CIAC 2010, Rome, Italy, May 26-28, 2010. Proceedings*, pages 167–179, 2010.

52 Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

53 Carsten Thomassen. On the presence of disjoint subgraphs of a certain type. *Journal of Graph Theory*, 12(1):101–111, 1988.

54 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *VLDB*, pages 82–94, 1981.

How to prove security of communication protocols? A discussion on the soundness of formal models w.r.t. computational ones.*

Hubert Comon-Lundh[1] and Véronique Cortier[2]

1 **ENS Cachan& CNRS & INRIA Saclay Ile-de-France**
2 **LORIA, CNRS**

Abstract

Security protocols are short programs that aim at securing communication over a public network. Their design is known to be error-prone with flaws found years later. That is why they deserve a careful security analysis, with rigorous proofs. Two main lines of research have been (independently) developed to analyse the security of protocols. On the one hand, formal methods provide with symbolic models and often automatic proofs. On the other hand, cryptographic models propose a tighter modeling but proofs are more difficult to write and to check. An approach developed during the last decade consists in bridging the two approaches, showing that symbolic models are *sound* w.r.t. symbolic ones, yielding strong security guarantees using automatic tools. These results have been developed for several cryptographic primitives (e.g. symmetric and asymmetric encryption, signatures, hash) and security properties.

While proving soundness of symbolic models is a very promising approach, several technical details are often not satisfactory. Focusing on symmetric encryption, we describe the difficulties and limitations of the available results.

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

Keywords and phrases Verification, security, cryptography

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.29

1 Introduction

Security protocols aim at securing communications over public networks. They are typically designed for bank transfers over the Internet, establishing private channels, or authenticating remote sites. They are also used in more recent applications such as e-voting procedures. Depending on the application, they are supposed to ensure security properties such as confidentiality, privacy or authentication, even when the network is (at least partially) controlled by malicious users, who may intercept, forge and send new messages. While the specification of such protocols is usually short and rather natural, designing a secure protocol is notoriously difficult and flaws may be found several years later. A famous example is the "man-in-the-middle" attack found by G. Lowe against the Needham-Schroder public key protocol [41]. A more recent example is the flaw discovered in Gmail (and now fixed) by Armando *et. al.* [9].

* This work has been supported by the ANR-07-SeSur-002 AVOTÉ project.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 29–44
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

During the two last decades, formal methods have demonstrated their usefulness when designing and analyzing security protocols. They indeed provide with rigorous frameworks and techniques that allow to discover new flaws. For instance, the two previously mentioned flaws have been discovered while trying to prove the security of the protocol in a formal setting. Following the seminal work of Dolev and Yao [33], many techniques have been developed for analysing the security of protocols, often automatically. For example, the AVISPA platform [8] and the ProVerif tool [20] are both efficient and practical tools for automatically proving security properties or finding bugs if any. The security of protocols is undecidable in general [34]. Checking the secrecy and authentication-like properties is however NP-complete when the number of sessions is fixed [44]. Several extensions have been designed, considering more security properties or more security primitives [2, 25, 28, 24, 37]. Bruno Blanchet has developed an (incomplete) procedure based on clause resolution [19] for analyzing protocols for an unbounded number of sessions. All these approaches rely on a common representation for messages: they are symbolically modeled by *terms* where each function symbol represents a cryptographic primitive, some of their algebraic properties being reflected in an equational theory. Then protocols are modeled using or adapting existing frameworks such as fragments of logic, process algebras or constraint systems.

While the symbolic approaches were successful in finding attacks, the security proofs in these models are questionable, because of the level of abstraction: most cryptographic details are ignored. This might be a problem: for instance, it is shown in [45] that a protocol can be proved in a formal, symbolic, model, while there is an attack, that also exploits some finer details of the actual implementation of the encryption scheme. In contrast, cryptographic models are more accurate: the security of protocols is based on the security of the underlying primitives, which in turn is proved assuming the hardness of various computational tasks such as factoring or computing discrete logarithms. The messages are bitstrings. The proofs in the computational model imply strong guarantees (security holds in the presence of an arbitrary probabilistic polynomial-time adversary). However, security reductions for even moderately-sized protocols become extremely long, difficult, and tedious. Recently, a significant research effort [6, 43, 13, 15, 11, 26] has been directed towards bridging the gap between the symbolic and the cryptographic approaches. Such *soundness* results typically show that, under reasonable cryptographic assumptions such as IND-CCA2 for the encryption scheme, proofs in symbolic models directly imply proofs in the more detailed cryptographic models. These approaches are very promising: they allow to reconcile two distinct and independently developed views for modeling and analysing security protocols. Second and more importantly, they allow to obtain the best of the two worlds: strong security guarantees through the simpler symbolic models, that are amenable to automatic proofs.

However, such soundness results also assume many other properties regarding the implementation or even regarding the key infrastructure. In this paper, we discuss these usually under-looked assumptions, pointing the limitations of current results. In particular, we provide with several protocols (counter) examples, for which IND-CCA2 does not imply the security, as soon as a malicious user may chose its own keys at its will. These examples show that standard symbolic models are not sound w.r.t. cryptographic ones when using symmetric encryption. We also discuss how to symbolically represent the length of messages and what are the implications on the implementation. All these examples will be discussed within the the applied-pi calculus [3], but the counter-examples do not depend on this particular process algebra: the discussion will stay at a rather informal level and can be understood without familiarity with the applied-pi calculus.

Related work. Many soundness results have been established in various settings. We discuss some of them in Section 3. Fewer works are dedicated to the limitations. Backes and Pfitzmann have shown that primitives such as Exclusive Or or hash functions cannot be soundly abstracted in their simulatability library [14]. This is related to the impossibility of constructing some universally composable primitives [40]. This witnesses the difficulty of designing sound and accurate models for some primitives. [1] compares CryptoVerif [21], an automatic tool designed for performing proofs directly in the cryptographic model, and the use of soundness results, emphasizing the current limitations of the latter.

2 Setting

We recall here briefly part of the syntax and the operational semantics of the applied π-calculus of [3]. We are going to use a small fragment of this calculus for the formal definition of the protocols.

2.1 Syntax

In any symbolic model for security protocols, messages are modeled by *terms*, which are built on a set of function symbols Σ, that represent the cryptographic primitives (e.g. encryption, pairing, decryption). Given an infinite set $\mathcal{N}$ of *names* and an infinite set $\mathcal{X}$ of *variables*, $\mathcal{T}(\mathcal{N}, \mathcal{X})$ is the set of *terms*:

$$\begin{array}{lll}
s, t, u \ ::= & \text{terms} & \\
\quad x, y, z & \text{variable} & \\
\quad a, b, c, k, n, r & \text{name} & \\
\quad f(s_1, \ldots, s_k) & \text{function application} & f \in \Sigma \text{ and } k \text{ is the arity of } f.
\end{array}$$

Terms represent messages and names stand for (randomly) generated data. We assume the existence of a length function l, which is a Σ-morphism from $T(\mathcal{N})$ to $\mathbb{N}$.

In what follows, we will consider symmetric encryption and pairing. Let Σ_0 consist of the binary pairing $< \cdot, \cdot >$, the two associated projections π_1, π_2, the binary decryption dec and the ternary symbol $\{\cdot\}^{\cdot}_{\cdot}$ for symmetric encryption: $\{x\}^r_k$ stands for the encryption of x with the key k and the random r. Σ_0 also contains constants, in particular a constant 0^l of length l for every l.

The syntax of processes is displayed in Figure 1. In what follows, we restrict ourselves to processes with public channels: there is no restriction on name channel. We assume a set $\mathcal{P}$ of predicate symbols with an arity. Such a definition, as well as its operational semantics coincides with [3], except for one minor point introduced in [26]: we consider conditionals with arbitrary predicates. This leaves some flexibility in modeling various levels of assumptions on the cryptographic primitives.

In what follows, we may use expressions of the form $\mathsf{let} \ \ldots \ \mathsf{in} \ \ldots$ as a syntactic sugar to help readability.

2.2 Operational semantics

We briefly recall the operational semantics of the applied pi-calculus (see [3, 26] for details). E is a set of equations on the signature Σ, defining an equivalence relation $=_E$ on $\mathcal{T}(\mathcal{N})$, which is closed under context. $=_E$ is meant to capture several representations of the same message. This yields a quotient algebra $\mathcal{T}(\mathcal{N})/=_E$, representing the messages. Predicate symbols are interpreted as relations over $\mathcal{T}(\mathcal{N})/=_E$. This yields a structure $\mathcal{M}$.

Φ_1, Φ_2 `::=`	conditions
$p(s_1, \ldots, s_n)$	predicate application
$\Phi_1 \wedge \Phi_2$	conjunction
P, Q, R `::=`	processes
$c(x).P$	input
$\bar{c}(s).P$	output
$\mathbf{0}$	terminated process
$P \parallel Q$	parallel composition
$!P$	replication
$(\nu\alpha)P$	restriction
`if` Φ `then` P `else` Q	conditional

Figure 1 Syntax of processes

In what follows, we will consider the equational theory E_0 on Σ_0 defined by the equations corresponding to encryption and pairing:

$$\mathsf{dec}(\{x\}^z_y, y) = x \qquad \pi_1(< x, y >) = x \qquad \pi_2(< x, y >) = y$$

These equations can be oriented, yielding a convergent rewrite system: every term s has a unique normal form $s\downarrow$.

We also consider the following predicates introduced in [26].

- M checks that a term is well formed. Formally, M is unary and holds on a (ground) term s iff $s\downarrow$ does not contain any projection nor decryption symbols and for any $\{u\}^r_v$ subterm of s, v and r must be names. This forbids compound keys for instance.
- EQ checks the equality of well-formed terms. EQ is binary and holds on s, t iff $M(s), M(t)$ and $s\downarrow = t\downarrow$: this is a strict interpretation of equality.
- $\mathsf{P}_{\mathsf{samekey}}$ is binary and holds on ciphertexts using the same encryption key: $\mathcal{M} \models \mathsf{P}_{\mathsf{samekey}}(s, t)$ iff $\exists k, u, v, r, r'.EQ(s, \{u\}^r_k) \wedge EQ(t, \{v\}^{r'}_k)$.
- EL is binary and holds on s, t iff $M(s), M(t)$ and s, t have the same length.

▶ **Example 2.1.** The Wide Mouth Frog [22] is a simple protocol where a server transmits a session key K_{ab} from an agent A to an agent B. This toy example is also used in [1] as a case study for both CryptoVerif and soundness techniques. For the sake of illustration, we propose here a flawed version of this protocol.

$$\begin{array}{lcl} A \rightarrow S & : & A, B, \{N_a, K_{ab}\}_{K_{as}} \\ S \rightarrow B & : & A, \{N_s, K_{ab}\}_{K_{bs}} \end{array}$$

The server is assumed to share long-term secret keys with each agent. For example, K_{as} denotes the long-term key between A and the server. In this protocol, the agent A establishes a freshly generated key K_{ab} with B, using the server for securely transmitting the key to B.

A session l_a of role A played by agent a with key k_{as} can be modeled by the process

$$A(a, b, k_{as}, l_a) \overset{\text{def}}{=} (\nu r, n_a)\ \overline{\mathsf{c_{out}}}(< l_a, < a, < b, \{< n_a, k_{ab} >\}^r_{k_{as}} >>>) \cdot \mathbf{0}$$

Similarly a session of role S played for agents a, b with corresponding keys k_{as} and k_{bs}, can be modeled by

$$
\begin{aligned}
&S(a, b, k_{as}, k_{bs}, l_s) \overset{\text{def}}{=} (\nu n_s, r)\ \mathsf{c_{in}}(x).\ \mathtt{if}\ EQ(\pi_1(x), l_s)\ \mathtt{then}\ \mathsf{let}\, y = \pi_2(\mathsf{dec}(\pi_2(\pi_2(\pi_2(x))), k_{as}))\ \mathsf{in}\\
&\mathtt{if}\ \ \pi_1(\pi_2(x)) = a\ \wedge\ \pi_1(\pi_2(\pi_2(x)) = b\ \wedge\ M(y)\ \mathtt{then}\\
&\ \overline{\mathsf{c_{out}}}(< l_s, < a, \{< n_s, y >\}^r_{k_{bs}} > >) \cdot \mathbf{0}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \mathtt{else}\ \overline{\mathsf{c_{out}}}(\bot) \cdot \mathbf{0}\ \mathtt{else}\ \overline{\mathsf{c_{out}}}(\bot) \cdot \mathbf{0}
\end{aligned}
$$

where l_s is the session identifier of the process.

Then an unbounded number of sessions of this protocol, in which A plays a (with b) and s plays S (with a, b and also with b, c) can be represented by the following process

$$
\begin{aligned}
P_{\mathsf{ex}} = \nu(k_{as}, k_{bs})\ \ (\ \ &!((\nu k_{ab}, l_a)\overline{\mathsf{c_{out}}}(l_a).A(a, b, k_{as}, l_a, r))\\
&\parallel\ !((\nu l_s)\overline{\mathsf{c_{out}}}(l_s).S(a, b, k_{as}, k_{bs}, l_s))\ \parallel\ !((\nu l_s)\overline{\mathsf{c_{out}}}(l_s).S(a, c, k_{as}, k_{cs}, l_s))\ \ )
\end{aligned}
$$

To reflect the fact that c is a dishonest identity, its long-term key k_{cs} shared with the server does not appear under a restriction and is therefore known to an attacker.

The environment is modeled through *evaluation context*, that is a process $C = (\nu\overline{\alpha})([\cdot] \| P)$ where P is a process. We write $C[Q]$ for $(\nu\overline{\alpha})(Q \parallel P)$. A context (resp. a process) C is *closed* when it has no free variables (there might be free names).

Possible evolutions of processes are captured by the relation $\rightarrow$, which is the smallest relation, compatible with the process algebra and such that:

$$
\begin{array}{rrcll}
\text{(Com)} & c(x).P \parallel \overline{c}(s).Q & \rightarrow & \{x \mapsto s\} \parallel P \parallel Q & \\
\text{(Cond1)} & \mathtt{if}\ \Phi\ \mathtt{then}\ P\ \mathtt{else}\ Q & \rightarrow & P & \text{if } \mathcal{M} \models \Phi\\
\text{(Cond2)} & \mathtt{if}\ \Phi\ \mathtt{then}\ P\ \mathtt{else}\ Q & \rightarrow & Q & \text{if } \mathcal{M} \not\models \Phi
\end{array}
$$

$\xrightarrow{*}$ is the smallest transitive relation on processes containing $\rightarrow$ and some structural equivalence (e.g. reflecting the associativity and commutativity of the composition operator $\|$) and closed by application of contexts.

▶ **Example 2.2.** Continuing Example 2.1, we show an attack, that allows an attacker to learn k_{ab}, the key exchanged between a and b. Indeed, an attacker can listen to the first message $< l_a, < a, < b, \{< n_a, k_{ab} >\}^{r_1}_{k_{as}} >>>$ and replace it with $< l_a, < a, < c, \{< n_a, k_{ab} >\}^{r_1}_{k_{as}} >>>$. Thus the server would think that a wishes to transmit her key k_{ab} to c. Therefore it would reply with $< a, \{< n_s, k_{ab} >\}^{r_2}_{k_{cs}} >$. The attacker can then very easily decrypt the message and learn K_{ab}. This attack corresponds to the context

$$
\begin{aligned}
&C_{\mathsf{attack}} \overset{\text{def}}{=} [\cdot] \parallel \mathsf{c_{out}}(x_{l_a}).\mathsf{c_{out}}(x_{l_s}).\mathsf{c_{out}}(x_{m_a}).\ //\text{listens to sessions ids and the first message}\\
&\ \mathsf{let}\ y = \pi_2(\pi_2(x_{m_a}))\ \mathsf{in}\\
&\overline{\mathsf{c_{in}}}(< x_{l_s}, < a, < c, y >>>).\ //\text{replays the message, with } b \text{ replaced by } c\\
&\mathsf{c_{out}}(x_{m_s}).\ //\text{listens to the server's reply}\\
&\qquad\qquad\qquad\qquad \mathsf{let}\ y' = \mathsf{dec}(\pi_2(\pi_2(x_{m_s})), k_{cs})\ \mathsf{in}\ \overline{\mathsf{c_{in}}}(\pi_2(y')).0\ //\text{outputs the secret}
\end{aligned}
$$

Then the attack is reflected by the transitions $C_{\mathsf{attack}}[P_{\mathsf{ex}}] \xrightarrow{*} \overline{\mathsf{c_{out}}}(k_{ab}) \parallel Q$ for some process Q, yielding the publication of the confidential key k_{ab}.

2.3 Observational equivalence

Observational equivalence is useful to describe many properties such as confidentiality or authentication as exemplified in [5]. It is also crucial for specifying privacy related properties as needed in the context of electronic voting protocols [32].

▶ **Definition 2.3.** The *observational equivalence relation* $\sim_o$ is the largest symmetric relation $\mathcal{S}$ on closed extended processes such that $A\mathcal{S}B$ implies:

1. if, for some context C, term s and process A', $A \overset{*}{\rightarrow} C[\overline{c}(s) \cdot A']$ then for some context C', term s' and process B', $B \overset{*}{\rightarrow} C'[\overline{c}(s') \cdot B']$.
2. if $A \overset{*}{\rightarrow} A'$ then, for some B', $B \overset{*}{\rightarrow} B'$ and $A'\mathcal{S}B'$
3. $C[A]\mathcal{S}C[B]$ for all closed evaluation contexts C

▶ **Example 2.4** (Group signature)**.** The security of group signature has been defined in [7]. It intuitively ensures that an attacker should not be able to distinguish two signatures performed with two distinct identities when they belong to the same group. It can be modeled as observational equivalence as follows. Let $P(x, i)$ be the protocol for signing message x with identity i. Let $P_0 = c(y).P(\pi_1(y), \pi_1(\pi_2(y)))$ and $P_1 = c(y).P(\pi_1(y), \pi_2(\pi_2(y)))$. Intuitively, the adversary will send $< m, < i_0, i_1 >>$ where m is a message to be signed and i_0, i_1 are two identities. P_0 signs m with i_0 while P_1 signs m with i_1. Then P preserves anonymity iff $P_0 \sim_o P_1$.

2.4 Computational interpretation

We assume given an encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$ where $\mathcal{G}$ is the generating function for keys, $\mathcal{E}$ is the encryption function and $\mathcal{D}$ the decryption function. We also assume given a pairing function. The encryption, decryption, and pairing functions and their corresponding projectors form respectively the computational interpretation of the symbols $\{\cdot\}, \mathsf{dec}, <, >, \pi_1, \pi_2$. We assume that the decryption and projection functions return an error message $\perp$ when they fail. Then, given an interpretation τ of names as bitstrings, $[\![\cdot]\!]_\tau$ is the (unique) Σ-morphism extending τ to $T(\mathcal{N})$; $[\![t]\!]_\tau$ is the *computational interpretation* of t. When τ is randomly drawn, according to a distribution that depends on a security parameter η, we may write $[\![t]\!]_\eta$ for the corresponding distribution and $[\![t]\!]$ for the corresponding family of distributions. Then here are possible interpretations of the predicates:

- $[\![M]\!]$ is the set of bitstrings, which are distinct from $\perp$. Intuitively $[\![M]\!]$ implements M if the encryption scheme is *confusion-free* (a consequence of INT-CTXT [42]).
- $[\![EQ]\!]$ is the set of pairs of identical bitstrings, which are distinct from $\perp$. It is an implementation of EQ as soon as $[\![M]\!]$ implements M.
- $[\![\mathsf{P}_{\mathsf{samekey}}]\!]$ is the set of pairs of bitstrings that have the same encryption tag.
- $[\![EL]\!]$ is the set of pairs of bitstrings of same length.

Processes can also be interpreted as communicating Turing machines. Such machines have been introduced in [16, 38] for modeling communicating systems. They are probabilistic Turing machines with input/output tapes. Those tapes are intuitively used for reading and sending messages. We will not describe them here and we refer to [26] for more details.

Now, given a process P without replication, one can interpret it as a (polynomial time) communicating Turing machine. The computational interpretation of P is denoted by $[\![P]\!]$ and is intuitively defined by applying the computational counterpart of each function and predicate symbols. Then the replicated process $!P$ can also be interpreted by letting the adversary play with as many copies of $[\![P]\!]$ as he wants.

Indistinguishability. In computational models, security properties are often stated as *indistinguishability* of games. Two families of machines are indistinguishable if an adversary cannot tell them apart except with non negligible probability.

A function $f : \mathbb{N} \to \mathbb{N}$ is *negligible* if, for every polynomial P, $\exists N \in \mathbb{N}, \forall \eta > N, f(\eta) < \frac{1}{P(\eta)}$. We write $\mathbf{Pr}\{x : P(x)\}$ the probability of event $P(x)$ when the sample x is drawn according to an appropriate distribution (the key distribution or the uniform distribution; this is kept implicit).

▶ **Definition 2.5.** Two environments $\mathcal{F}$ and $\mathcal{F}'$ are *indistinguishable*, denoted by $\mathcal{F} \approx \mathcal{F}'$, if, for every polynomial time communicating Turing Machine A (i.e. for any attacker),

$$|\mathbf{Pr}\{\overline{r}, r : (\mathcal{F}(\overline{r}) \parallel A(r))(0^\eta) = 1\} - \qquad \mathbf{Pr}\{\overline{r}, r : (\mathcal{F}'(\overline{r}) \parallel A(r))(0^\eta) = 1\}|$$

is negligible. $\overline{r}$ is the sequence of random inputs of the machines in $\mathcal{F}$ (resp. $\mathcal{F}'$), including keys. r is the random input of the attacker.

For example, anonymity of group signatures as discussed in Example 2.4 is defined in [7] through the following game: the adversary chooses a message m and two identities i_0 and i_1. Then in $\mathcal{F}_0$, the machines sign m with identity i_0 while in $\mathcal{F}_1$, the machines sign m with identity i_1. Then the anonymity is defined by $\mathcal{F}_0 \approx \mathcal{F}_1$. Note that, for $i = 1, 2$, $\mathcal{F}_i$ can be defined as $[\![P_i]\!]$, implementation of the process P_i of the Example 2.4.

More generally, security properties can be defined by specifying the ideal behavior P_{ideal} of a protocol P and requiring that the two protocols are indistinguishable. For example, in [4], authenticity is defined through the specification of a process where the party B magically received the message sent by the party A. This process should be indistinguishable from the initial one.

3 Soundness results

Computational models are much more detailed than symbolic ones. In particular, the adversary is very general as it can be any (polynomial) communicating Turing machine. Despite the important difference between symbolic and computational models, it is possible to show that symbolic models are *sound* w.r.t. computational ones.

3.1 A brief survey

There is a huge amount of work on simulatability/universal composability, especially the work of Backes *et. al.* and Canetti [23, 13, 15, 11]. When the ideal functionality is the symbolic version of the protocol, then the black-box simulatability implies the trace mapping property [11], therefore showing a safe abstraction. Such results can be applied to trace properties such as authentication but not to indistinguishability. In a recent paper [12], Backes and Unruh show that the whole applied-pi calculus can be embedded in CoSP, a framework in which they prove soundness of public-key encryption and digital signatures, again for trace properties.

Besides [26], which we discuss in more detail below, one of the only results that prove soundness for indistinguishability properties is [39], for some specific properties (see the end of section 4 for more details).

In a series of papers starting with Micciancio and Warinschi [43] and continued with e.g. [31, 36], the authors show trace mapping properties: for some selected primitives (public-key encryption and signatures in the above-cited papers) they show that a computational trace is an instance of a symbolic trace, with overwhelming probability. But, again, this does not show that indistinguishability properties can be soundly abstracted, except for the special case of computational secrecy that can be handled in [31] and also in [29] for hash functions in the random oracle model.

We refer to [30] for a more complete survey of soundness results.

3.2 Observational equivalence implies indistinguishability

The main result of [26] consists in establishing that observational equivalence implies indistinguishability:

▶ **Theorem 3.1.** *Let P_1 and P_2 be two simple processes such that each P_i admits a key hierarchy. Assume that the encryption scheme is joint IND-CPA and INT-CTXT. Then $P_1 \sim_o P_2$ implies that $[\![P_1]\!] \approx [\![P_2]\!]$.*

This result assumes some hypotheses, some of which are explicitly stated above and informally discussed below (the reader is referred to [26] for the full details). There are additional assumptions, that are discussed in more details in the next section.

Simple processes are a fragment (introduced in [26]) of the applied-pi calculus. It intuitively consists of parallel composition of (possibly replicated) *basic processes*, that do not involve replication, parallel composition or else branches. Simple processes capture most protocols without else branch, for an unbounded number of sessions. For example, the process P_{ex} introduced in Example 2.1 is a basic process.

The most annoying restriction is the absence of conditional branching. Ongoing works should overcome this limitation, at the price of some additional computational assumptions. But the extension to the full applied π-calculus is really challenging, because of possible restrictions on channel names. Such restrictions indeed allow "private" computations, of which an attacker only observes the computing time (which is not part of the model).

Key hierarchy ensures that no key cycles can be produced on honest keys, even with the interaction of the adversary. This hypothesis is needed because current security assumptions such as IND-CPA do not support key cycles (most encryption schemes are not provably secure in the presence of key cycles). In [26], it is assumed that there exists a strict ordering on key such that no key encrypts a greater key.

Checking such conditions, for any possible interaction with the attacker, is in general undecidable, though a proof can be found in many practical cases. (And it becomes decidable when there is no replication [27]).

No dynamic corruption assumes that keys are either immediately revealed (e.g. corrupted keys) or remain secret. Showing a soundness result in case of dynamic key corruption is a challenging open question, that might require stronger assumptions on the encryption scheme.

IND-CPA and INT-CTXT are standard security assumptions on encryption schemes. The IND-CPA assumption intuitively ensures that an attacker cannot distinguish the encryption of any message with an encryption of zeros of the same length. INT-CTXT ensures that an adversary cannot produce a valid ciphertext without having the encryption key. IND-CPA and INT-CTXT are standard security assumptions [18].

4 Current limitations

We discuss in this section the additional assumptions of Theorem 3.1. Let us emphasize that such assumptions are not specific to this result: other soundness results have similar restrictions and/or provide with a weaker result.

4.1 Parsing

Parsing the bitstrings into terms is used in the proof of the soundness results; this function has actually to computable in polynomial time, since this is part the construction of a Turing machine used in a reduction. Therefore, Theorem 3.1 assumes that the pairing, key generation and encryption functions add a typing tag (which can be changed by the attacker), that indicates which operator has been used and further includes which key is used in case of encryption. This can be achieved by assuming that a symmetric key k consists of two parts (k_1, k_2), k_1 being generated by some standard key generation algorithm and k_2 selected at random. Then one encrypts with k_1 and tags the ciphertext with k_2.

These parsing assumptions are easy to implement and do not restrict the computational power of an adversary. Adding tags can only add more security to the protocol. However, current implementations of protocols do not follow these typing hypotheses, in particular regarding the encryption. Therefore Theorem 3.1 requires a reasonable but non standard and slightly heavy implementation in order to be applicable.

The parsing assumption might be not necessary. There are ongoing works trying to drop it.

4.2 Length function

As explained in Section 2, Theorem 3.1 assumes the existence of a length function l, which is a morphism from $T(\mathcal{N})$ to $\mathbb{N}$. This length function is needed to distinguish between ciphertexts of different lengths. For example, the two ciphertext $\{< n_1, n_2 >\}_k$ and $\{n_1\}_k$ should be distinguishable while $\{< n_1, n_2 >\}_k$ and $\{< n_1, n_1 >\}_k$ should not. There are however cases where it is unclear whether the ciphertexts should be distinguishable or not:

$$\nu k.\overline{\mathsf{c_{out}}}(\{< n_1, n_2 >\}_k) \overset{?}{\sim}_o \nu k.\overline{\mathsf{c_{out}}}(\{\{n_1\}_k\}_k).$$

Whether these two ciphertexts are distinguishable typically depends on the implementation and the security parameter: their implementation may (or not) yield bitstrings of equal length. However, for a soundness result, we need to distinguish (or not) these ciphertexts *independently* of the implementation.
In other words, if we let length be the length of a bitstring, we (roughly) need the equivalence:

$$l(s) = l(t) \qquad \text{iff} \qquad \forall \tau.\ \mathsf{length}([\![s]\!]_\tau) = \mathsf{length}([\![t]\!]_\tau)$$

This requires some length-regularity of the cryptographic primitives. But even more, this requires length to be homogenous w.r.t. the security parameter η. To see this, consider the case where length is an affine morphism:

$$\begin{array}{rcl}
\mathsf{length}([\![\{t_1\}_k^r]\!]_\tau) & = & \mathsf{length}([\![t_1]\!]_\tau) + \gamma \times \eta + \alpha \\
\mathsf{length}([\![< t_1, t_2 >]\!]_\tau) & = & \mathsf{length}([\![t_1]\!]_\tau) + \mathsf{length}([\![t_2]\!]_\tau) + \beta \\
\mathsf{length}(\tau(n)) & = & \delta \times \eta
\end{array}$$

where β cannot be null since some bits are needed to mark the separators between the two strings $[\![t_1]\!]$ and $[\![t_2]\!]$. (Also, $\gamma, \delta > 0$.)

Now, if we consider an arbitrary term t, $\mathsf{length}([\![t]\!]_\tau) = n_1 \times \gamma \times \eta + n_1 \times \alpha + n_2 \times \beta + n_3 \times \delta \times \eta$. $\frac{\mathsf{length}([\![s]\!]_\tau)}{\mathsf{length}([\![t]\!]_\tau)}$ must be independent of η, hence there must exist $\alpha', \beta' \in \mathbb{N}, \beta' > 0$ such that $\alpha = \alpha' \times \eta$ and $\beta = \beta' \times \eta$.

This implies in particular that the pairing function always adds η bits (or a greater multiple of η) to the bitstrings. Similarly, a ciphertext should be the size of its plaintext plus a number of bits which is the a multiple of η.

While it is possible to design an implementation that achieves such constraints, this is not always the case in practice and it may yield a heavy implementation, in particular in conjunction with the parsing assumptions. Moreover, on the symbolic side, adding a length function raises non trivial decidability issues.

Adding a symbolic length function is needed for proving indistinguishability as illustrated by the former examples. It is worth noticing that several soundness results such as [13, 15, 31, 29, 12] do not need to consider a length function. The reason is that they focus on *trace properties* such as authentication but they cannot considered indistinguishability-based properties (except computational secrecy for some of them).

4.3 Dishonest keys

Theorem 3.1 assumes the adversary only uses correctly generated keys. In particular, the adversary cannot choose his keys at its will, depending on the observed messages. The parties are supposed to check that the keys they are using have been properly generated. The assumption could be achieved by assuming that keys are provided by a trusted server that properly generates keys together with a certificate. Then when a party receives a key, it would check that it comes with a valid certificate, guaranteeing that the key has been issued by the server. Of course, the adversary could obtain from the server as many valid keys as he wants.

However, this assumption is strong compared to usual implementation of symmetric keys and it is probably the less realistic assumptions among those needed for Theorem 3.1. We discuss alternative assumptions at the end of this section. It is worth noticing that in all soundness results for asymmetric encryption, it is also assumed that the adversary only uses correctly generated keys. Such an assumption is more realistic in an asymmetric setting as a server could certify public keys. However, this does not reflect most current implementations for public key infrastructure, where agents generate their keys on their own.

We now explain why such an assumption is needed to obtain soundness.The intuitive reason is that IND-CCA does not provide any guarantee on the encryption scheme when keys are dishonestly generated.

▶ **Example 4.1.** Consider the following protocol. A sends out a message of the form $\{c\}_{K_{ab}}$ where c is a constant. This can be formally represented by the process

$$A = (\nu r)\overline{\mathsf{c_{out}}}(< c, \{c\}^r_{k_{ab}} >).\mathbf{0}$$

Then B expects a key y and a message of the form $\{\{b\}_y\}_{K_{ab}}$ where b is the identity of B, in which case, it sends out a secret s (or goes in a bad state).

$$\begin{array}{rll} A & \to B : & (\nu r)\ c, \{c\}^r_{K_{ab}} \\ B : k, \{\{b\}_k\}_{K_{ab}} & \to A : & s \end{array}$$

This can be formally modeled by the process

$$B = \mathsf{c_{in}}(z).\ \texttt{if}\ EQ(b, \mathsf{dec}(\mathsf{dec}(\pi_2(z), k_{ab}), \pi_1(z))\ \texttt{then}\ \overline{\mathsf{c_{out}}}(s)\ \texttt{else}\ \mathbf{0}$$

Then symbolically, the process $(\nu k_{ab})(\nu s)A\|B$ never emits s. However, a computational adversary can forge a key k such that any bitstring can be successfully decrypted to b using k. In particular, at the computational level, we have $\mathsf{dec}(c, k) = b$. Thus by sending $< k, \{c\}^r_{k_{ab}} >$ to B, the adversary would obtain the secret s.

This is due to the fact that security of encryption schemes only provides guarantees on *properly generated* keys. More precisely, given an IND-CCA2 (authenticated) encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$, it is easy to build another IND-CCA2 (authenticated) encryption scheme $(\mathcal{G}', \mathcal{E}', \mathcal{D}')$, which allows to mount the previous attack: consider $\mathcal{G}' = 0 \cdot \mathcal{G}$ (all honest keys begin with the bit 0), $\mathcal{E}'(m, i.k) = \mathcal{E}(m, k)$ for $i \in \{0, 1\}$ and $\mathcal{D}'$ defined as follows:

- $\mathcal{D}'(c, k) = \mathcal{D}(c, k')$ if $k = 0 \cdot k'$
- $\mathcal{D}'(c, k) = k'$ if $k = 1 \cdot k'$

It is easy to check that $(\mathcal{G}', \mathcal{E}', \mathcal{D}')$ remains IND-CCA2 and allows an adversary to choose a key that decrypts to anything he wants.

To capture this kind of computational attacks, an idea (from M. Backes [10]) is to enrich the symbolic setting with a rule that allows an intruder, given a ciphertext c and a message m, to forge a key such that c decrypts to m. This could be modeled e.g. by adding a functional symbol fakekey of arity 2 together with the equation

$$\mathsf{dec}(x, \mathsf{fakekey}(x, y)) = y$$

Going back to Example 4.1, this would allow a symbolic intruder to send the message $< \mathsf{fakekey}(c, b), \{c\}^r_{k_{ab}} >$ to the B process and the process $(\nu k_{ab})(\nu s)A\|B$ would emit s.

This solution appears however to be insufficient to cover the next examples.

▶ **Example 4.2** (hidden cyphertext)**.** The same kind of attacks can be mounted even when the ciphertext is unknown to the adversary. We consider a protocol where A sends $<< A, k >, \{\{k'\}^{r'}_k\}^r_{k_{ab}} >$ where k and k' are freshly generated keys. B recovers k' and sends it encrypted with k_{ab}. In case A receives her name encrypted with k_{ab}, A emits a secret s.

$$\begin{array}{rll} A & \to B: & (\nu k, k', r_1, r_2)\ A, k, \{\{k'\}^{r_1}_k\}^{r_2}_{K_{ab}} \\ B & \to A: & (\nu r_3)\ \{k'\}^{r_3}_{K_{ab}} \\ A : \{A\}_{K_{ab}} & \to B: & s \end{array}$$

Then symbolically, the process $(\nu k_{ab})(\nu s)A\|B$ would never emit s while again, a computational adversary can forge a key k such that any bitstring can be successfully decrypted to a using k.

This attack could be captured, allowing the forged key to be independent of the ciphertext. This can be modeled by the equation

$$\mathsf{dec}(x, \mathsf{fakekey}(y)) = y$$

where fakekey is now a primitive of arity 1. Some attacks may however require the decryption to depend from the cyphertext as shown in the next example.

▶ **Example 4.3** (simultaneous cyphertexts)**.** Consider the following protocol where A sends to B p cyphertexts $c_1, \ldots, c_p$. Then B encrypts all ciphertexts with a shared key k_{ab} together with a fresh value n_b and commits to p other nonces $N_1, \ldots, N_p$. Then A simply forwards the cyphertext together with a fresh key k. Then B checks whether each cyphertext c_i decrypts to N_i using the key k received from A, in which case he sends out a secret s.

$$\begin{array}{rll} A & \to B: & c_1, \ldots, c_p \\ B & \to A: & \{N_b, c_1, \ldots, c_p\}_{K_{ab}}, N_1, \ldots, N_p \\ A & \to B: & \{N_b, c_1, \ldots, c_p\}_{K_{ab}}, k \\ B: \{N_b, \{N_1\}_k, \ldots, \{N_p\}_k\}_{K_{ab}}, k & \to A: & s \end{array}$$

Then symbolically, the process $(\nu k_{ab})(\nu s)A\|B$ would never emit s since the N_i are generated after having received the c_i and the nonce N_b protects the protocol from replay attacks. However, having seen the c_i and the N_i, a computational adversary can forge a key k such that each bitstring c_i can be successfully decrypted to N_i using k. More precisely, given an IND-CCA2 (authenticated) encryption scheme $(\mathcal{G}, \mathcal{E}, \mathcal{D})$, it is easy to build another IND-CCA2 (authenticated) encryption scheme $(\mathcal{G}', \mathcal{E}', \mathcal{D}')$, which allows to mount the previous attack. Indeed, consider $\mathcal{G}' = 0 \cdot \mathcal{G}$ (all honest keys begin with the bit 0), $\mathcal{E}'(m, i.k) = \mathcal{E}(m, k)$ for $i \in \{0, 1\}$ and $\mathcal{D}'$ defined as follows:

- $\mathcal{D}'(c, k) = \mathcal{D}(c, k')$ if $k = 0 \cdot k'$
- $\mathcal{D}'(c, k) = n$ if $k = 1 \cdot c_1, n_1, \cdots c, n \cdots c_p, n_p$
- $\mathcal{D}'(c, k) = \perp$ otherwise

For dishonest keys, the decryption function $\mathcal{D}'(c, k)$ searches for c in k and outputs the following component when c is found in k. It is easy to check that $(\mathcal{G}', \mathcal{E}', \mathcal{D}')$ remains IND-CCA2 and allows an adversary to chose a key that decrypts to anything he wants, but with different possible outputs depending on the ciphertext.

To capture this attack, we need to consider a symbol of arity $2p$ for any p and an equation of the form

$$\mathsf{dec}(x_i, \mathsf{fakekey}(x_1, \ldots, x_p, y_1, \ldots, y_p)) = y_i$$

But this is still not be sufficient as the outcome may also depend on the ciphertext that is under decryption and on public data. Intuitively, decrypting with an adversarial key may produce a function depending on the underlying plaintext and on any previously known data.

Related work. To our best understanding of [13], these examples seem to form counter-examples of the soundness results for symmetric encryption as presented in [13]. An implicit assumption that solves this issue [10] consists in forbidding dishonest keys to be used for encryption or decryption (the simulator would stop as soon as it received a dishonest keys). As a consequence, only protocols using keys as nonces could be proved secure.

The only work overcoming this limitation in a realistic way is the work of Kuesters and Tuengerthal [39] where the authors show computational soundness for key exchange protocols with symmetric encryption, without restricting key generation for the adversary. Instead, they assume that sessions identifiers are added to plaintexts before encryption. This assumption is non standard but achievable. It would however not be sufficient in general as shown by the examples. In their case, such an assumption suffices because the result is tailored to key exchange protocols and realization of a certain key exchange functionality.

5 Conclusion

Among all the limitations we discuss in this paper, the main one is to consider only honestly generated keys (or a certifying infrastructure), which is completely unrealistic. There are (at least) two main ways to overcome this assumption. A first possibility, already sketched in the paper, consists in enriching the symbolic model by letting the adversary create new symbolic equalities when building new (dishonest) keys. In this way, many protocols should still be provably secure under the IND-CCA assumption, yet benefiting from a symbolic setting for writing the proof.

A second option is to seek for stronger security assumptions by further requesting non-malleability. The idea is that a ciphertext should not be opened to a different plaintext, even when using dishonest keys. This could be achieved by adding a commitment to the encryption scheme [35].

However all these limitations also demonstrate that it is difficult to make symbolic and computational models coincide. Even for standard security primitives, soundness results are very strong since they provide with a generic security proof for *any* possible protocol (contrary to CryptoVerif). For primitives with many algebraic properties like Exclusive Or or modular exponentiation, the gap between symbolic and computation models is even larger and would require a lot of efforts.

We still believe that computational proofs could benefit from the simplicity of symbolic models, yielding automated proofs. An alternative approach to soundness results could consist in computing, out of a given protocol, the minimal computational hypotheses needed for its security. This is for example the approach explored in [17], though the symbolic model is still very complex.

Acknowledgement

We wish to thank Michael Backes and Dominique Unruh for very helpful explanations on their work and David Galindo for fruitful discussion on non-malleable encryption schemes.

References

1 M. Abadi, B. Blanchet, and H. Comon-Lundh. Models and proofs of protocol security: A progress report. In A. Bouajjani and O. Maler, editors, *Proceedings of the 21st International Conference on Computer Aided Verification (CAV'09)*, volume 5643 of *Lecture Notes in Computer Science*, pages 35–49, Grenoble, France, June-July 2009. Springer.

2 M. Abadi and V. Cortier. Deciding knowledge in security protocols under equational theories. *Theoretical Computer Science*, 387(1-2):2–32, November 2006.

3 M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *Proc. of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, January 2001.

4 M. Abadi and A. Gordon. A calculus for cryptographic protocols: the spi calculus. *Information and Computation*, 148(1), 1999.

5 M. Abadi and A. D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Proc. of the 4th ACM Conference on Computer and Communications Security (CCS'97)*, pages 36–47. ACM Press, 1997.

6 M. Abadi and P. Rogaway. Reconciling two views of cryptography. In *Proc. of the International Conference on Theoretical Computer Science (IFIP TCS2000)*, pages 3–22, August 2000.

7 M. Abdalla and B. Warinschi. On the minimal assumptions of group signature schemes. In *6th International Conference on Information and Communication Security*, pages 1–13, 2004.

8 A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. Hankes Drielsma, P.-C. Héam, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb,

M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The AVISPA Tool for the automated validation of internet security protocols and applications. In K. Etessami and S. Rajamani, editors, *17th International Conference on Computer Aided Verification, CAV'2005*, volume 3576 of *Lecture Notes in Computer Science*, pages 281–285, Edinburgh, Scotland, 2005. Springer.

9 A. Armando, R. Carbone, L. Compagna, J. Cuéllar, and M. L. Tobarra. Formal analysis of saml 2.0 web browser single sign-on: breaking the saml-based single sign-on for google apps. In *6th ACM Workshop on Formal Methods in Security Engineering (FMSE 2008)*, pages 1–10, Alexandria, VA, USA, 2008.

10 M. Backes. Private communication, 2007.

11 M. Backes, M. Dürmuth, and R. Küsters. On simulatability soundness and mapping soundness of symbolic cryptography. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, 2007.

12 M. Backes, D. Hofheinz, and D. Unruh. CoSP a general framework for computational soundness proofs. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS 2009)*, pages 66–78, 2009.

13 M. Backes and B. Pfitzmann. Symmetric encryption in a simulatable Dolev-Yao style cryptographic library. In *Proc. 17th IEEE Computer Science Foundations Workshop (CSFW'04)*, pages 204–218, 2004.

14 M. Backes and B. Pfitzmann. Limits of the Cryptographic Realization of Dolev-Yao-style XOR and Dolev-Yao-Style Hash Functions. In *Proc. 10th European Symposium on Research in Computer Security (ESORICS'05)*, Lecture Notes in Computer Science, pages 336–354, 2005.

15 M. Backes and B. Pfitzmann. Relating cryptographic und symbolic key secrecy. In *26th IEEE Symposium on Security and Privacy*, pages 171–182, Oakland, CA, 2005.

16 M. Backes, B. Pfitzmann, and M. Waidner. The reactive simulatability (RSIM) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

17 G. Bana, K. Hasebe, and M. Okada. Secrecy-oriented first-order logical analysis of cryptographic protocols. Cryptology ePrint Archive, Report 2010/080, 2010. `http://eprint.iacr.org/`.

18 M. Bellare and C. Namprempre. Authenticated encryption: relations among notions and analysis of the generic composition paradigm. In *Advances in Cryptology (ASIACRYPT 2000)*, volume 1976 of *Lecture Notes in Computer Science*, pages 531–545, 2000.

19 B. Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proc. of the 14th Computer Security Foundations Workshop (CSFW'01)*. IEEE Computer Society Press, June 2001.

20 B. Blanchet. An automatic security protocol verifier based on resolution theorem proving (invited tutorial). In *20th International Conference on Automated Deduction (CADE-20)*, July 2005.

21 B. Blanchet. A computationally sound mechanized prover for security protocols. *IEEE Transactions on Dependable and Secure Computing*, 5(4):193–207, Oct.–Dec. 2008. Special issue IEEE Symposium on Security and Privacy 2006. Electronic version available at http://doi.ieeecomputersociety.org/10.1109/TDSC.2007.1005.

22 M. Burrows, M. Abadi, and R. Needham. A logic of authentication. In *Proc. of the Royal Society*, volume 426 of *Series A*, pages 233–271. 1989. Also appeared as SRC Research Report 39 and, in a shortened form, in ACM Transactions on Computer Systems 8, 1 (February 1990), 18-36.

23 R. Canetti and M. Fischlin. Universally composable commitments. In *CRYPTO 2001*, pages 19–40, Santa Barbara, California, 2001.

24 Y. Chevalier, R. Küsters, M. Rusinowitch, and M. Turuani. An NP Decision Procedure for Protocol Insecurity with XOR. *Theoretical Computer Science*, 338(1-3):247–274, June 2005.

25 Y. Chevalier, R. Küsters, M. Rusinowitch, M. Turuani, and L. Vigneron. Deciding the security of protocols with Diffie-Hellman exponentiation and product in exponents. In *Proc. of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, 2003.

26 H. Comon-Lundh and V. Cortier. Computational soundness of observational equivalence. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08)*. ACM Press, 2008.

27 H. Comon-Lundh, V. Cortier, and E. Zalinescu. Deciding security properties for cryptographic protocols. Application to key cycles. *ACM Transactions on Computational Logic (TOCL)*, 11(4):496–520, 2010.

28 H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science (LICS'03)*, pages 271–280, Ottawa, Canada, June 2003. IEEE Computer Society Press.

29 V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. In N. Garg and S. Arun-Kumar, editors, *Proceedings of the 26th Conference on Fundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 176–187, Kolkata, India, December 2006. Springer.

30 V. Cortier, S. Kremer, and B. Warinschi. A Survey of Symbolic Methods in Computational Analysis of Cryptographic Systems. *Journal of Automated Reasoning (JAR)*, 2010.

31 V. Cortier and B. Warinschi. Computationally Sound, Automated Proofs for Security Protocols. In *Proc. 14th European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 157–171, Edinburgh, U.K, April 2005. Springer.

32 S. Delaune, S. Kremer, and M. D. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Computer Security Foundations Workshop (CSFW'06)*, pages 28–39, 2006.

33 D. Dolev and A. Yao. On the security of public key protocols. In *Proc. of the 22nd Symp. on Foundations of ComputerScience*, pages 350–357. IEEE Computer Society Press, 1981.

34 N. Durgin, P. Lincoln, J. Mitchell, and A. Scedrov. Undecidability of bounded security protocols. In *Proc. of the Workshop on Formal Methods and Security Protocols*, 1999.

35 D. Galindo, F. D. Garcia, and P. Van Rossum. Computational soundness of non-malleable commitments. In *Proceedings of the 4th international conference on Information security practice and experience*, ISPEC'08, pages 361–376. Springer-Verlag, 2008.

36 R. Janvier, Y. Lakhnech, and L. Mazaré. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *European Symposium on Programming (ESOP'05)*, volume 3444 of *Lecture Notes in Computer Science*, pages 172–185. Springer, 2005.

37 R. Küsters and T. Truderung. On the automatic analysis of recursive security protocols with xor. In *Proceedings of the 24th International Symposium on Theoretical Aspects of Computer Science (STACS'07)*, volume 4393 of *LNCS*, Aachen, Germany, 2007. Springer.

38 R. Küsters and M. Tuengerthal. Joint state theorems for public-key encryption and digital signature functionalities with local computations. In *Computer Security Foundations (CSF'08)*, 2008.

39 R. Küsters and M. Tuengerthal. Computational Soundness for Key Exchange Protocols with Symmetric Encryption. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS 2009)*, pages 91–100. ACM Press, 2009.

40 Y. Lindell. General composition and universal composition in secure multiparty computation. In *Proc. 44th IEEE Symp. Foundations of Computer Science (FOCS)*, 2003.

41 G. Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In T. Margaria and B. Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS'96)*, volume 1055 of *LNCS*, pages 147–166. Springer-Verlag, march 1996.

42 D. Micciancio and B. Warinschi. Completeness theorems for the Abadi-Rogaway language of encrypted expressions. *Journal of Computer Security*, 2004.

43 D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Theory of Cryptography Conference (TCC 2004)*, volume 2951 of *Lecture Notes in Computer Science*, pages 133–151, Cambridge, MA, USA, February 2004. Springer-Verlag.

44 M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete. *Theoretical Computer Science*, 299:451–475, April 2003.

45 B. Warinschi. A computational analysis of the needham-schroeder protocol. In *16th Computer security foundation workshop (CSFW)*, pages 248–262. IEEE, 2003.

Local dependency dynamic programming in the presence of memory faults

Saverio Caminiti, Irene Finocchi, and Emanuele G. Fusco

Department of Computer Science, *Sapienza* University of Rome
Via Salaria 113, 00198 Rome, Italy
{caminiti,finocchi,fusco}@di.uniroma1.it

Abstract

We investigate the design of dynamic programming algorithms in unreliable memories, i.e., in the presence of faults that may arbitrarily corrupt memory locations during the algorithm execution. As a main result, we devise a general resilient framework that can be applied to all local dependency dynamic programming problems, where updates to entries in the auxiliary table are determined by the contents of neighboring cells. Consider, as an example, the computation of the edit distance between two strings of length n and m. We prove that, for any arbitrarily small constant $\varepsilon \in (0, 1]$ and $n \geq m$, this problem can be solved correctly with high probability in $O\left(nm + \alpha\delta^{1+\varepsilon}\right)$ worst-case time and $O(nm + n\delta)$ space, when up to δ memory faults can be inserted by an adversary with unbounded computational power and $\alpha \leq \delta$ is the actual number of faults occurring during the computation. We also show that an optimal edit sequence can be constructed in additional time $O\left(n\delta + \alpha\delta^{1+\varepsilon}\right)$. It follows that our resilient algorithms match the running time and space usage of the standard non-resilient implementations while tolerating almost linearly-many faults.

1998 ACM Subject Classification B.8 [Performance and reliability]; F.2 [Analysis of algorithms and problem complexity]; I.2.8 [Dynamic programming].

Keywords and phrases Unreliable memories, fault-tolerant algorithms, local dependency dynamic programming, edit distance.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.45

1 Introduction

Dynamic random access memories (DRAM) are susceptible to errors, where the logical state of one or multiple bits is read differently from how it was last written. Such errors may be due either to hardware problems or to transient electronic noises [14]. A recent large-scale study of DRAM memory errors reports data collected in the field from Google's server fleet over a period of nearly 2.5 years [20] observing DRAM error rates that are orders of magnitude higher than previously reported in laboratory conditions. As an example, a cluster of 1000 computers with 4 gigabytes per node can experience one bit error every three seconds, with each node experiencing an error every 40 minutes. If errors are not corrected, they can lead to a machine crash or to applications using corrupted data. Silent data corruptions are a major concern in the reliability of modern storage systems, since even a few of them may be harmful to the correctness and performance of software. To cope with this, a recent trend is to design applications that are more tolerant to faults: this "robustification" of software involves re-writing it so that dealing with faults simply causes the execution to take longer. Unfortunately, most algorithms and data structures are far from being robust: since the contents of memory locations are supposed not to change throughout the execution unless they are explicitly written by the program, wrong steps may be taken upon reading corrupted

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 45–56
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

values, yielding unpredictable results. Coping with memory faults appears to be of particular importance for all those applications handling massive data sets, for long-living processes, for safety critical applications in avionics systems, and for cryptographic protocols, that can be compromised by fault-based attacks that work by inducing very timing-precise bit flips.

Related work. Algorithmic research related to memory errors spans more than thirty years. Starting from the "twenty questions game" posed by Rényi and Ulam in the late 70's, many results have been obtained in the liar model: see, e.g., the extensive survey in [18]. More recently, sorting and selection have been studied in the "just noticeable difference model", where the outcome of comparisons is unpredictable if the compared values are within a fixed threshold [1]. All these works typically assume transient comparator failures, but no corruption of data. Destructive faults have been first investigated in the context of fault-tolerant sorting networks [17], and many subsequent works have focused on the design of resilient data structures in a variety of (hardly comparable) models. Pointer-based data structures are the subject of [2] and error-correcting data structures for fundamental problems related to membership have been presented in [7, 9]. The more restrictive problem of checking (but not recovering) the behavior of large data structures that reside in an unreliable memory has also received considerable attention [3, 8].

A variety of resilient algorithms have been designed in the faulty-memory random access machine (*faulty RAM*) introduced in [12], where an adversary can corrupt at most δ memory cells of a large unreliable memory during the execution of an algorithm. The algorithms can exploit knowledge of δ, which is a parameter of the model, and the existence of a constant number of incorruptible registers, but do not require error detection capabilities. Resiliency is achieved if a problem is solved correctly (at least) on the set of uncorrupted values. This relaxed definition of correctness fits naturally sorting and searching problems addressed so far in this model [10], as well as the design of resilient data structures such as dictionaries [11] and priority queues [15]. As an example, given a set of n values, it is possible to sort correctly the subset of uncorrupted values in a comparison-based model using optimal $\Theta(n \log n)$ time when $\delta = O(\sqrt{n})$ [10]. Resilient counters in the faulty RAM model are described in [6], showing different tradeoffs between the time for incrementing a counter and its additive error. Motivated by the impact of memory errors on applications operating with massive data sets, the connection between fault-tolerance and I/O-efficiency is investigated in [5], providing the first external-memory algorithms resilient to memory faults.

Our results. In spite of the wealth of results summarized above, it remains an open question whether powerful algorithmic techniques such as those based on dynamic programming can be made to work in the presence of faults. This has been regarded as an elusive goal for many years in a variety of faulty memory models. In this paper we provide the first positive answers to this question by showing how to implement a large class of dynamic programming algorithms resiliently in unreliable memories. We consider the faulty RAM model introduced in [12] and we illustrate our techniques using as a case study the problem of computing the edit distance between two strings of length n and m. A simple-minded resilient implementation of the standard dynamic programming algorithm for edit distance could be based on replicating all data (string symbols and table values) $2\delta + 1$ times. By applying majority techniques, this would allow to tolerate up to δ faults at the cost of a multiplicative $\Theta(\delta)$ overhead on both space usage and running time. Hence, only a *constant* number of faults could be tolerated while maintaining the standard $O(nm)$ time bound. In contrast, we devise algorithms that preserve this bound while tolerating, with high probability, up to an almost *linear* number of faults. We will prove that this is nearly optimal. More formally, we show that the edit distance between two strings of length n and m can be correctly computed, with high probability, in $O\left(nm + \alpha\delta^{1+\varepsilon}\right)$ worst-case time and $O(nm + n\delta)$ space, when $\alpha \leq \delta$ faults occur during the computation, $n \geq m$, and ε is an arbitrarily small constant in $(0, 1]$. Our algorithms exploit knowledge of δ and only a constant number of private memory words. If the private memory can be enlarged to $O(\log \delta)$ words, the fault-dependent additive term in the running time becomes $O(\alpha\delta)$. The framework we provide is general enough to be applied to all local dependency dynamic programming problems, where updates to entries in the auxiliary table are determined by the contents of neighboring cells: this is a significant class of problems that includes, e.g., longest common subsequence and many sequence alignment problems in bioinformatics. Our framework can be made deterministic, yielding an algorithm that tolerates a logarithmic number of faults, and can be extended to incorporate well-known optimizations of dynamic programming, such as Hirschberg's space-saving technique [13] and Ukkonen's distance-sensitive algorithm [21]. Due to the lack of space, some proofs and details are omitted.

Techniques. Our resilient implementation does not rely on any cryptographic assumption. Instead, it hinges upon a novel combination of majority techniques (which are a typical but expensive error correction method), read and write Karp-Rabin fingerprints (to detect faults), and an asymmetric, hierarchical decomposition of the dynamic programming table into rectangular slices of height δ and decreasing width (to bound the cost of error recovery). We remark that, although fingerprints have been successfully used in the context of checking the correctness of data structures, they alone are not powerful enough in the faulty RAM model, where the goal is to recover the computation when a fault is detected without restarting it from scratch. Hence, to obtain the $O\left(\alpha\delta^{1+\varepsilon}\right)$ additive term in the running time, we exploit as a main ingredient a hierarchy of $O(1/\varepsilon)$ levels of data replication. At all levels, except for the last one, data are stored *semi-resiliently* in the unreliable memory, by replicating each variable $o(\delta)$ times. Notice that semi-resilient data could be corrupted by the adversary, but at the cost of a large number of faults: this will allow us to amortize the cost of a slice recomputation (semi-resilient variables need to be appropriately "refreshed" upon detection of faults, so that the cost of a slice recomputation can always be charged to distinct faults). $O(1/\varepsilon)$ *long-distance fingerprints* stored in safe memory make it possible to backtrack the computation, at any time, to a checkpoint that is safe with high probability. Combining semi-resiliency with refreshing and long-distance fingerprints allows us to guarantee the correctness of the table computation while bounding the error recovery cost.

A different technique must be used during the traceback process that computes an optimal solution from its optimal value: at this point, long-distance fingerprints are no longer available, and thus we have no guarantee that semi-resilient variables are correct. To overcome this issue, we proceed incrementally in $O(1/\varepsilon)$ passes: at each pass, either we increase our confidence that the computed path is correct, or we are guaranteed that the adversary has introduced a large number of faults.

2 Preliminaries

We assume a unit cost RAM with wordsize w. We distinguish between *unreliable*, *safe*, and *private* memory. Up to δ unreliable memory words may be corrupted during the execution of an algorithm by an adaptive adversary with unlimited computational power. We denote by $\alpha \leq \delta$ the actual number of faults occurring during the computation. No error-detection mechanism is provided. We have $O(1)$ safe memory words that the adversary can read but not overwrite: without this assumption, no reliable computation would be possible [12].

Similarly to [3, 10, 11], we also assume $O(1)$ private memory words, that the adversary cannot even read: this is necessary to prevent the adversary from discovering random bits used by the algorithms.

Resilient variables. A *resilient variable* x consists of $2\delta + 1$ copies of a standard variable [11]. A *reliable write* operation on x means assigning the same value to each copy. Similarly, a *reliable read* means calculating the majority value, which is correct since at most δ copies can be corrupted. This can be done in $\Theta(\delta)$ time with the majority algorithm in [4], which scans the $2\delta + 1$ values keeping a single majority candidate and a counter in safe memory. Throughout the paper we will also make use of *r-resilient* variables (with $r < \delta$), which consist of $2r + 1$ copies of a standard variable. A r-resilient read operation on an (at least) r-resilient variable is obtained by computing the majority value on $2r + 1$ copies. Notice that a r-resilient variable can be corrupted by the adversary, but at the cost of at least $r + 1$ faults.

Generation of random primes. Random primes are usually generated by selecting a number uniformly at random and testing it for primality with, e.g., the Miller-Rabin test [19]. If the test is successful, the selected number is returned, otherwise a new candidate is selected and the process is iterated. The Miller-Rabin test has one-sided error: it can output *prime* for a composite number with a provably small probability. We keep this scheme almost unchanged, except for bounding the number of iterations so as to avoid having an expected running time. Although the probability of failure in our case does not uniquely depend on the Miller-Rabin test, it is not difficult to prove that this probability remains small and that the algorithm can be executed in our model:

▶ **Lemma 1.** *For any constants $\gamma, c > 0$, it is possible to independently select α (not necessarily distinct) prime numbers in $I = [n^{c-1}, n^c]$, uniformly at random, with error probability bounded by α/n^γ. Each prime selection requires time polylogarithmic in n using a constant number of memory words.*

3 An $O\left(nm + \alpha\delta^2\right)$ algorithm for edit distance

Given two strings $X = x_1 \cdots x_n$ and $Y = y_1 \cdots y_m$ over a finite alphabet Σ, the edit distance (a.k.a. Levenshtein distance) between X and Y is the number of edit operations (insertions, deletions, or character substitutions) required to transform X into Y. Let $e_{i,j}$, for $0 \le i \le n$ and $0 \le j \le m$, be the edit distance between prefix $x_1 \cdots x_i$ of string X and prefix $y_1 \cdots y_j$ of string Y (the prefix is empty if $i = 0$ or $j = 0$). Values $e_{i,j}$ are defined as follows:

$$e_{i,j} := \begin{cases} e_{i-1,j-1} & \text{if } i,j > 0 \text{ and } x_i = y_j \\ 1 + \min\left\{e_{i-1,j}, e_{i,j-1}, e_{i-1,j-1}\right\} & \text{if } i,j > 0 \text{ and } x_i \neq y_j \end{cases} \tag{1}$$

where $e_{i,0} = i$, $e_{0,j} = j$, and $e_{n,m}$ represents the edit distance of X and Y. The standard dynamic programming algorithm stores values $e_{i,j}$, for $i, j > 0$, in a $n \times m$ table M whose entries can be computed, e.g., in column-major order in $\Theta(nm)$ time. While calculating the edit distance only requires to keep in memory two table columns, an optimal edit sequence can be obtained by a tracing back process that reads table M backward: in this case, the space usage of the standard implementation is $\Theta(nm)$.

When executed in faulty memories, such a dynamic programming algorithm could provide a wrong result, since undetected memory faults could corrupt either the input strings or the values stored in table M and could be easily propagated to $M[n, m]$. In the rest of this section we describe a basic resilient edit distance algorithm (RED) with running time

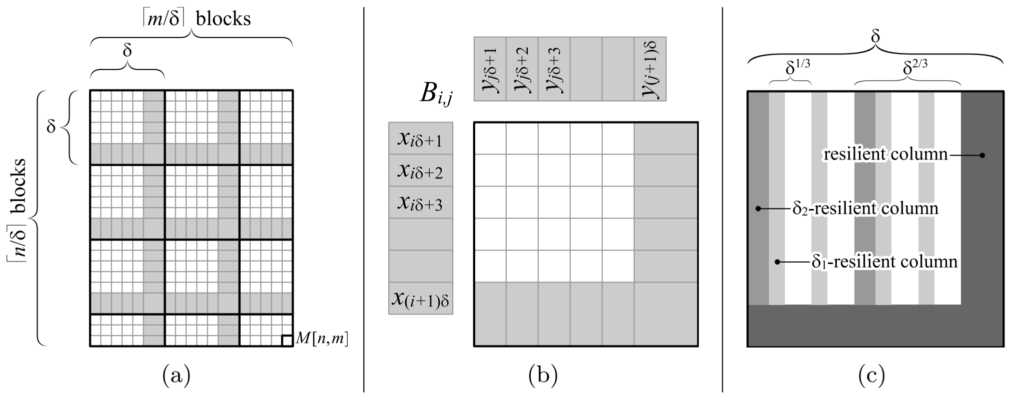


Figure 1 a) Table decomposition and resilient block boundaries (gray color) when $n \geq m \geq \delta$; b) block computation with unreliable input; c) hierarchical table decomposition for $k = 3$.

$O(nm + \alpha\delta^2)$. In Section 4 we will show how to decrease the fault-dependent additive term in the running time. W.l.o.g., throughout the paper we assume that $n \geq m$; we also assume that n and $|\Sigma|$ fit into a memory word of size w.

Algorithm RED mimics the behavior of the standard non-resilient dynamic programming approach, performing additional work in order to cope with memory faults. During the computation of table M, we compute fingerprints that allow us to determine whether some memory fault occurred in a given set of memory words. Once detected, a fault should not force us to recompute the entire table: this would result in a $\Theta(\delta)$ multiplicative overhead on the running time in the worst case. Hence, we divide the table in blocks and we write the boundaries of the blocks reliably in the unreliable memory. Upon detection of a failure, we recompute only the current block. We now describe the table decomposition into blocks, the computation of each block, and the usage of fingerprints to detect faults. We also describe how to handle faulty input strings.

Table decomposition. The $n \times m$ table M is split into square *blocks* of side length δ (see Figure 1a). Algorithm RED populates table M block by block, considering blocks in column-major order. The last row and column of each block are written reliably in the unreliable memory, using $2\delta + 1$ memory words for each value as described in Section 2. It follows that each block of δ^2 values takes roughly $5\delta^2$ memory words. Blocks are smaller (and not necessarily square) on the boundaries, whenever n or m are not divisible by δ: in this case the last row and/or column may not be written reliably.

Block computation. Let $B_{i,j}$ be an internal block (boundary blocks can be treated similarly). Entries of $B_{i,j}$ are processed in column-major order. The first column of $B_{i,j}$ is computed reliably: this is possible because row δ of block $B_{i-1,j}$, column δ of $B_{i,j-1}$, and entry $B_{i-1,j-1}[\delta, \delta]$ are written reliably in the unreliable memory. During the computation of column 1, a fingerprint φ_1 is calculated (details are given below). Let us now consider a generic column k, for $k > 1$. Each value $v_1, \ldots, v_\delta$ in column k is written unreliably in the faulty memory as soon as it is computed, except for the values in row δ and in column δ that are written reliably. While scanning column k top-down, the algorithm also computes two fingerprints: a fingerprint φ_k that is a function of values $v_1, \ldots, v_\delta$ *written* to column k, and a fingerprint $\overline{\varphi}_{k-1}$ that is a function of values *read* from column $k - 1$. The two fingerprints, together with the fingerprint φ_{k-1} previously computed during the calculation of column $k - 1$, are stored in the private memory. When column k is completed, algorithm

RED compares fingerprints φ_{k-1} and $\overline{\varphi}_{k-1}$: we call this a *fingerprint test.* If the fingerprints match, both of them are discarded and the computation of column $k+1$ begins. Otherwise, a memory fault has been detected: in this case the computation of the block is restarted.

Fault detection. We use Karp-Rabin fingerprints [16] to detect faults. The fingerprint for column k is defined as $\varphi_k = v_1 \circ v_2 \circ \ldots \circ v_\delta \bmod p$, where the δ values v_h of column k are considered as bit strings of length equal to the word size w and symbol $\circ$ denotes string concatenation. The concatenation $v_1 \circ v_2 \circ \ldots \circ v_\delta$ is thus an integer, upper bounded by $2^{w\delta}$, corresponding to the binary representation of the entire column k. Number p is a sufficiently large prime, chosen uniformly at random at the beginning of the execution of the algorithm and after each fault detection. Using logical shifts and Horner's rule, each fingerprint can be incrementally computed while generating the values v_h in time $\Theta(\delta)$. All computations related to the fingerprints are performed in $O(1)$ private memory words, so that no information regarding the prime number p is revealed to the adversary.

Handling the input strings. We assume each symbol in strings X and Y to be written reliably in the unreliable memory after its first reading (it is clearly impossible to detect a fault corrupting an input symbol before it is read for the first time). In order to compute matrix M in $O(nm)$ time, each comparison of the input symbols required by Equation 1 must be performed in constant (amortized) time. Let $B_{i,j}$ be a $\delta \times \delta$ block. The input values involved in the computation of $B_{i,j}$ are $x_{i\delta+1}, \ldots, x_{(i+1)\delta}$ and $y_{j\delta+1}, \ldots, y_{(j+1)\delta}$ (see Figure 1b). Consider a column k of block $B_{i,j}$. Character $y_{j\delta+k}$ is the only character of string Y needed to compute the values in column k: we read $y_{j\delta+k}$ reliably and amortize this $\Theta(\delta)$ operation on the cost of computing the δ values of column k. Conversely, all characters $x_{i\delta+1}, \ldots, x_{(i+1)\delta}$ are required to compute each column of $B_{i,j}$, and we cannot afford reading each of them reliably δ times. These input characters are thus read reliably once, while computing the first column of the block, producing a fingerprint φ_x that is kept in the private memory. While processing a column $k > 1$, values $x_{i\delta+1}, \ldots, x_{(i+1)\delta}$ are read unreliably (i.e., considering only one copy) and a fingerprint $\overline{\varphi}_{x,k}$ is computed. If fingerprint $\overline{\varphi}_{x,k}$ is different from φ_x, a memory fault has been detected: the resilient variables $x_{i\delta+1}, \ldots, x_{(i+1)\delta}$ are refreshed and the entire block is recomputed from scratch.

We now analyze algorithm RED, focusing first on correctness.

▶ **Lemma 2.** *For any constant $\beta > 0$, algorithm* RED *is correct with probability larger than $1 - 1/n^\beta$, when the upper bound δ on the number of memory faults is polynomial in n.*

Proof. Assume that the algorithm fails either when a composite number is generated instead of a prime, or when a fingerprint test does not detect a memory fault. This is an overestimation of the actual probability of error. Let B be a block that gets corrupted during its own computation. Assume for the time being that values in the boundaries of its neighboring blocks are correct. Then, the values written to column 1 of block B are also correct, because the neighboring entries used to compute column 1 and all input symbols involved are read reliably. By applying standard techniques, it is possible to prove that the probability that a fingerprint test does not detect a memory fault during the computation of B is at most $(\log n)/(\sigma n^{c-1}) < 1/(\sigma n^{c-2})$, for some constant $\sigma > 0$.

Now consider a game with two players. The game is divided into rounds. At each round player 1 (the algorithm) chooses uniformly at random a prime $p \in I$ and player 2 (the adversary) chooses a number $\mu \leq 2^{wd}$. If p divides μ, then player 2 wins, otherwise the next round begins. Player 1 wins if player 2 does not win in α rounds. This game models the behavior of algorithm RED, provided that no composite number is generated instead of a prime. Namely, the probability for algorithm RED of being correct is lower bounded by the probability for player 1 of winning the game.

Let p_i and μ_i be the numbers chosen by the two players at round i. Let D_i be the event "player 2 does not win at round i". If player 2 did not win in rounds $1, \dots, i-1$, the probability of D_i equals the probability that p_i does not divide μ_i. From the discussion above, we have $Pr\left\{D_i | \bigcap_{j=1}^{i-1} D_j\right\} \geq 1 - 1/(\sigma n^{c-2})$. The probability that player 1 wins is equal to $Pr\left\{\bigcap_{i=1}^{\alpha} D_i\right\}$, which is at least $1 - \alpha/(\sigma n^{c-2})$ by the chain rule of conditional probability.

We conclude by taking into account the probability for algorithm RED of generating at some round a composite number instead of a prime. By Lemma 1, the probability that all the α numbers are prime is at least $1 - \alpha/n^{\gamma}$, for any constant $\gamma > 0$. Hence, algorithm RED is correct with probability larger than or equal to $(1 - \alpha/(\sigma n^{c-2}))(1 - \alpha/n^{\gamma})$. Since $\alpha \leq \delta$ is polynomial in n, by appropriately choosing values c and γ the correctness probability can be made larger than $1 - 1/n^{\beta}$, for any constant $\beta > 0$. ◀

It is not difficult to see that the space usage of algorithm RED is $\Theta(nm)$ when $m = \Omega(\delta)$, and $\Theta(n\delta)$ otherwise. Lemma 3 addresses the running time of the algorithm.

▶ **Lemma 3.** *The worst-case running time of algorithm* RED *is $O(nm + \alpha\delta^2)$, where $\alpha \leq \delta$ is the actual number of memory faults occurring during the execution.*

Proof. Let us distinguish between *successful* and *unsuccessful block computations*. Unsuccessful block computations account for the time spent by the algorithm computing blocks that are then discarded due to the detection of a memory fault. This time also includes the generation of random primes, except for the first one. Successful block computations account for the remaining time, including the calculation of fingerprints.

Successful computations. Computing the first column of a block requires constantly-many reliable reads for each entry, i.e., $O(\delta^2)$ time. The same bound holds for the last column, that is written reliably. Computing any internal column requires instead $O(\delta)$ time, including the time to compute fingerprints incrementally. Since there are $O(\delta)$ columns in a block, the total time spent in a block is $O(\delta^2)$. The overall time for successful block computations is thus $O(nm)$, because the number of blocks is $\lceil n/\delta \rceil \times \lceil m/\delta \rceil$.

Unsuccessful computations. Each block recomputation is due to a fingerprint mismatch, that can only be caused by a memory fault (either in the matrix cells or in some input symbol from string X). Since all block cells are recomputed and, if necessary, the input symbols are refreshed reading their values reliably, each block recomputation can be charged to a distinct memory fault. It follows that at most α block computations can be discarded during the entire execution of algorithm RED. Refreshing δ input values and computing the block take time $O(\delta^2)$, which implies an overall time $O(\alpha\delta^2)$ for unsuccessful computations. The generation of (at most α) prime numbers does not affect this asymptotic running time (see Lemma 1). ◀

4 Error recovery via long distance fingerprints

Using a one-level decomposition of the dynamic programming table M into squares of side length δ yields an algorithm with an additive term $O(\alpha\delta^2)$ in the running time, due to recovery from errors (a single error determines the complete recomputation of a $\delta \times \delta$ block). In this section we show how to decrease this time to $O(\alpha\delta^{1+\varepsilon})$, for any arbitrarily small constant $\varepsilon \in (0, 1]$. The improved algorithm uses an asymmetric decomposition (see Figure 1c) and $k = \lceil 1/\varepsilon \rceil$ different *resiliency levels*. At each level $i \in [1, k]$, it relies on $\lceil \delta^{i/k} \rceil$-resilient variables. To simplify the notation, we define $\delta_i = \lceil \delta^{i/k} \rceil$.

Consider a given $\delta \times \delta$ block B. Every δ_i columns, we write a δ_i-resilient column (and all δ_j-resilient versions of this column for $j < i$). In particular, the last column of each internal block is written at all resiliency levels. The non-resilient columns of matrix M are regarded as having resiliency level 0. During the computation of block B, for each resiliency level i we keep (in the private memory) the fingerprint of the last δ_i-resilient column. These long distance fingerprints, similarly to those described in Section 3, are computed while writing column values. For each resiliency level, we independently select a prime number for computing the fingerprints.

Upon detection of a fault, error recovery is done starting from the last δ_1-resilient column. Values in this column are read by majority; read values are used to recompute the fingerprint at level 1 which is then compared with the one stored in the private memory. If these fingerprints do not match, the recovery starts again from resiliency level 2, i.e., from the last δ_2-resilient column. In general, a level i fingerprint mismatch induces a recovery starting from the last δ_{i+1}-resilient column. When a fingerprint mismatch arises at resiliency level i, we generate a new random prime for level i, we read by majority all values of the last δ_{i+1}-resilient column (i.e., we perform δ read operations at resiliency level $i + 1$), and we use these values to refresh all δ_j-resilient versions of this column, for $j \leq i$, recomputing their respective fingerprints.

Now consider the input symbols. As in Section 3, symbols from string Y are always read reliably, while symbols from X are read reliably only once, at the beginning of a block computation, and then verified by means of fingerprints. We store δ_i-resilient copies of the symbols in X at all resiliency levels. During the computation of a block, for each resiliency level (including level 0), we keep one fingerprint for the segment of X of length δ involved in that computation. All these fingerprints are obtained using independently selected prime numbers and are computed at the beginning of the block computation by reading reliably the input segment. Once a fingerprint mismatch on the input symbols is detected at level i, the δ_{i+1}-resilient copy of the input segment is used to refresh all copies at level $j \leq i$ (the previously computed fingerprint for resiliency level $i + 1$ allows it to check the correctness of the read values). A new random prime for level i is then selected and the fingerprints for all refreshed levels are recomputed. Notice that a fingerprint mismatch at level 0 may arise during block computation, while a mismatch at level $i > 0$ can only arise during error recovery. Once the input symbols are correctly refreshed, normal computation is resumed by recomputing the current column.

▶ **Theorem 4.** *Let ε be an arbitrarily small constant in $(0, 1]$. The edit distance between two strings of length n and m, with $n \geq m$, can be correctly computed, with high probability, in $O\left(nm + \alpha\delta^{1+\varepsilon}\right)$ worst-case time and $O(nm + n\delta)$ space, when δ is polynomial in n.*

Proof. The correctness of the improved version of algorithm RED follows from Lemma 2 (details are deferred to the extended version of this paper). Similarly to the proof of Lemma 3, we analyze the running time by distinguishing between successful and unsuccessful computations. The asymptotic running time of successful computations is not affected by the additional $O(1/\varepsilon)$ fingerprints and by the δ_i-resilient columns. Now consider the unsuccessful computations. Recovery at resiliency level i discards at most $\delta \times \delta^{i/k}$ entries of table M and requires computing $O(\delta)$ majority values. Each δ_i-resilient read takes time $O(\delta^{i/k})$, thus yielding total $O(\delta^{1+i/k})$ time. A recovery at resiliency level i is due to at least $\delta^{(i-1)/k} + 1$ errors, either on the input symbols or in table M. Errors can be propagated by the algorithm (during both refresh operations and forward block computations) only if a fingerprint test fails, which is a low probability event. Hence, a fingerprint mismatch at resiliency level $i-1$ may only arise if the majority value of some δ_{i-1}-resilient cell or input symbol has been corrupted by the adversary. This gives an amortized time per fault of $O(\delta^{1+1/k})$, which proves the theorem since $k = \lceil 1/\varepsilon \rceil$. ◀

Theorem 4 implies that, whenever $m = \Theta(n)$, algorithm RED matches the time and space complexity of the standard non-resilient implementation while tolerating almost linearly-many memory faults; specifically, up to $\delta = O\left(n^{2/(2+\varepsilon)}\right)$ faults. Notice that saving reliably the input strings requires time and space $\Omega(n\delta)$. Hence, any resilient algorithm which tolerates $\delta = \omega(n)$ memory faults must have time and space complexity $\omega(n^2)$. Hence, under the assumption that the time and space complexity of the standard non-resilient implementation cannot be exceeded, algorithm RED tolerates an almost optimal number of faults. If $O(\log \delta)$ private memory words are available (similarly to [3, 8]), the time bound given in Theorem 4 drops to $O(nm + \alpha\delta)$, thus allowing to tolerate an optimal linear number of faults.

5 Resilient traceback

Once table M has been computed, an optimal edit sequence transforming string X into string Y can be obtained by computing a traceback path from entry $M[n,m]$ to $M[0,0]$. The predecessor of an entry $M[i,j]$ on the traceback path can be any of the neighboring entries $M[i-1,j]$, $M[i,j-1]$, and $M[i-1,j-1]$, satisfying Equation 1. It will be convenient to assume that there is an arc from $M[i,j]$ to its predecessor: the cost of the arc is 0 if $x_i = y_j$, and 1 otherwise. We define the cost of a traceback path as the sum of the costs of its arcs: this corresponds to the edit distance of X and Y. We now describe how the traceback process can be made resilient.

The computation proceeds backward block by block, starting from cell $M[n,m]$. Within each block traversed by the traceback path, we compute the corresponding subsequence S of the whole edit sequence, writing it reliably. Data replication on the resilient block boundaries and on the input symbols allows, once S is computed, to check whether some error occurred during the backward computation. In order to recover from an error at this point, we could recompute first the block involved (by applying algorithm RED), and then subsequence S. This would result in an additive overhead $O(\alpha\delta^2)$ on the total running time. To bound this cost by $O(\alpha\delta^{1+\varepsilon})$, we do not stick at computing each subsequence S reliably since the beginning, but proceed incrementally starting from resiliency level 1 up to $k = \lceil 1/\varepsilon \rceil$. To this aim, we exploit the δ_i-resilient columns written during the forward computation of matrix M. The fact that column fingerprints are no longer available makes the task harder.

We regard each subsequence S as being divided into (at most) $\delta^{1/k}$ segments, computed at resiliency level $k-1$. This subdivision proceeds hierarchically, down to resiliency level 1. As a base step, segments at resiliency level 0 are computed from the cells of matrix M: these segments correspond to single arcs of the traceback path. A segment S_i, at resiliency level i, spans two δ_i-resilient columns (the right/left column, in some cases, can be replaced by the bottom/top resilient row of the block). S_i is computed by combining all the $\delta^{1/k}$ sub-segments at resiliency level $i-1$ in which S_i is logically divided. Sub-segments are read, proceeding right to left, δ_{i-1}-resiliently, and their soundness is verified against the corresponding input symbols, which are read δ_i-resiliently: we call this a *consistency check* (the δ_i-resilient reads on the input symbols from string Y are performed on the first $2\delta_i + 1$ elements of the δ-resilient copy of Y). During this process, S_i is also written δ_i-resiliently.

If a consistency check fails at a given cell c, the input symbols corresponding to the row and column of c are refreshed and, if either endpoint of S lies on a resilient row, the corresponding cell is also refreshed. The recovery then starts from the closest δ_i-resilient column to the left of c: all δ_j-resilient versions of this column, for $j < i$, are refreshed from the δ_i-resilient values (read by majority) and the block slice is recomputed by applying the improved version of algorithm RED. At the end of the slice computation, we check if the new values stored on the closest δ_i-resilient column to the right of c match the old ones: if this is not the case, recovery restarts at resiliency level $i+1$. At the end of the recovery phase, the computation of S_i restarts from sub-segments at resiliency level 1.

When the computation of a segment S_i is completed, the algorithm checks if the cost of the segment matches the difference between the cell values in matrix M corresponding to its endpoints. These cells lie on δ_i-resilient columns (or on resilient rows on the block boundaries) and their values are read δ_i-resiliently. Apart from refreshing the input values, upon detection of a mismatch in the edit sequence cost, recovery is performed as described above.

▶ **Theorem 5.** *Given table M computed by algorithm* RED*, an edit sequence of cost $M[n,m]$, if any, can be computed with high probability in time $O\left(n\delta + \alpha\delta^{1+\varepsilon}\right)$.*

Proof. The correctness of all edit operations is verified at all resiliency levels by consistency checks. Moreover, the edit cost of each segment is always verified against the edit distance. Memory faults from any resiliency level i are never propagated to higher levels of resiliency. Indeed, during error recovery, no δ_{i+j}-resilient value is modified starting from δ_i-resilient reads, for any $j \geq 0$. This implies that a segment at resiliency level i may be wrong only if at least $\delta^{i/k}+1$ memory faults occurred. Since the adversary can insert at most δ faults, the δ-resilient edit sequence, if constructed, is correct and has cost $M[n,m]$. This sequence may not be optimal or the traceback algorithm may not be able to reconstruct it only if a fingerprint test failed to detect a memory fault, which is a low probability event (see Lemma 2). We now focus on the running time, distinguishing between successful and unsuccessful segment computations.

Successful computation. The time spent to combine all δ_{k-1}-resilient segments is asymptotically higher than the time spent at all lower resiliency levels. This time is $O(n\delta)$, because the edit sequence traverses $O((n+m)/\delta)$ blocks, and each block costs time $O(\delta^2)$.

Unsuccessful computation. Consider a consistency check failure arising while computing a segment at resiliency level $i+1$. Such a failure is due to at least $\delta^{i/k}+1$ faults and costs $O(\delta^{1+(i+1)/k})$ time for recomputing $\delta \times \delta^{(i+1)/k}$ matrix cells. If the δ_{i+1}-resilient column used for recovery was correct, detected errors are removed from the matrix with high probability, with an amortized $O(\delta^{1+1/k})$ cost per memory fault. If the δ_{i+1}-resilient column used for recovery was corrupted, the adversary must have inserted at least $\delta^{(i+1)/k}+1$ faults and the recomputed cells of the matrix may still contain incorrect values after recovery. Two cases may happen: either the forward recomputation of the matrix slice finds an inconsistency with the following δ_{i+1}-resilient column, or no inconsistency is detected and a possibly wrong δ_{i+1}-resilient segment is computed. In both cases, the number of memory faults inserted by the adversary is large enough to obtain an amortized $O(\delta^{1+1/k})$ cost per fault, with recovery done at a higher resiliency level. ◀

6 Extensions

Reducing space. Hirschberg proposed a technique to compute an optimal edit sequence in time $O(nm)$ using only linear space [13]. In our model, $\Omega(n\delta)$ space is required for storing the input reliably. This is better than $\Theta(nm)$ when $\delta = o(m)$. We now show that this bound can be achieved by adapting Hirschberg's technique to work in faulty memories. Hirschberg's algorithm is recursive. In the first step, it computes the edit distances between the first half of string X and all prefixes of string Y, and between the remaining half of X reversed and all prefixes of Y reversed. It then finds an optimal point to split Y, constructs a segment of the optimal edit sequence, and recursively solves two smaller subproblems. We use algorithm RED to obtain the edit distances on the forward and reversed substrings: since no traceback is required, we discard a block as soon as all its neighboring blocks have been processed. The space usage is thus $O(n\delta)$. The split point can be computed reliably in $O(m\delta)$ time. Each recursive call pushes on the stack only $O(1)$ variables, that are stored and reloaded reliably. We end each branch of the recursion when the subproblem matrix has size bounded by $\delta \times \delta$, i.e., fits in a single block. It can be proved that this algorithm computes an optimal edit sequence resiliently in time $O(nm + \alpha\delta^{1+\varepsilon})$ and optimal space $\Theta(n\delta)$.

Taking advantage of string similarity. Ukkonen proved that an optimal edit sequence can be computed in time and space $O(e \min\{m, n\})$, where e is the edit distance between the input strings [21]. Since $e \geq n - m$, this improves over the standard dynamic programming algorithm only when $m = n - o(n)$, which implies $\min\{m, n\} = m = \Theta(n)$. The main idea is to assume that the edit distance e is upper bounded by a small value k and to compute only $\Theta(k)$ diagonals of matrix M. If no edit sequence of cost $\leq k$ exists, k is doubled and the computation is repeated. In the resilient implementation, we avoid considering blocks that have empty intersection with the set of diagonals that have to be computed in the current iteration. This results in a time and space complexity $O(n \max\{e, \delta\})$, matching the result from Ukkonen when $\delta = O(e)$. If $\delta = \omega(e)$, the $O(n\delta)$ time and space bounds match those required to handle the input reliably.

Local dependency dynamic programming. In the description of algorithm RED we exploit no specific properties of the edit distance problem. Instead, the analysis benefits from a few structural properties of the dynamic programming recurrence relation, that are also typical of many other problems. The techniques described in this paper can be applied to a variety of problems that can be solved via dynamic programming and, in particular, to local dependency dynamic programming problems, where each update to an entry in the auxiliary table is determined by the contents of the neighboring cells. The framework in which our technique can be applied successfully can be described as follows.

Let us consider a generic d-dimensional dynamic programming algorithm, for any constant $d \geq 2$. Assume that the problem input consist of d sequences $S_1, \ldots, S_d$, each of length n. The sequences are not necessarily distinct and the description can be easily generalized to deal with different lengths. The algorithm computes an auxiliary table M of dimension $(n+1)^d$. Each cell $M[i_1, \ldots, i_d]$, with $0 \leq i_1, \ldots, i_d \leq n$, is computed using a recurrence relation. In particular, if any index is equal to 0, then $M[i_1, \ldots, i_d]$ is initialized with a value that depends only on $i_1, \ldots, i_d$. Otherwise, $M[i_1, \ldots, i_d]$ is recursively computed from the values of the $2^d - 1$ neighboring cells, where a cell $M[j_1, \ldots, j_d]$ is a neighbor of a distinct cell $M[i_1, \ldots, i_d]$ if, for each dimension h, either $j_h = i_h - 1$ or $j_h = i_h$. Besides the neighboring cells, the computation of $M[i_1, \ldots, i_d]$ can also use d input symbols, i.e., the i_h-th symbol from sequence S_h for $1 \leq h \leq d$. We assume that the table is computed according to a fixed regular pattern (e.g., along rows, columns, or diagonals when $d = 2$), and that $M[n, \ldots, n]$ contains the solution. Using blocks of dimension δ^d, we can generalize our approach obtaining the following result:

▶ **Theorem 6.** *Let ε be an arbitrarily small constant in $(0, 1]$. A d-dimensional local dependency dynamic programming table M of size n^d can be correctly computed, with high* *probability, in $O(n^d + \alpha\delta^{d-1+\varepsilon})$ worst-case time and $O(n^d + n\delta)$ space, when δ is polynomial in n. Tracing back can be done with high probability in additional time $O(n\delta + \alpha\delta^{d-1+\varepsilon})$.*

This yields resilient algorithms for d-dimensional problems that have the same running time as the non-resilient implementations and can tolerate with high probability $O(n^{d/(d+\varepsilon)})$ memory faults.

References

1 M. Ajtai, V. Feldman, A. Hassidim, and J. Nelson. Sorting and selection with imprecise comparisons. In *ICALP (1)*, pages 37–48, 2009.

2 Y. Aumann and M. A. Bender. Fault tolerant data structures. In *FOCS*, pages 580–589, 1996.

3 M. Blum, W. S. Evans, P. Gemmell, S. Kannan, and M. Naor. Checking the correctness of memories. *Algorithmica*, 12(2–3):225–244, 1994.

4 R. S. Boyer and J. S. Moore. Mjrty: A fast majority vote algorithm. In *Automated Reasoning: Essays in Honor of Woody Bledsoe*, pages 105–118, 1991.

5 G. S. Brodal, A. G. Jørgensen, and T. Mølhave. Fault tolerant external memory algorithms. In *WADS*, pages 411–422, 2009.

6 G. S. Brodal, A. G. Jørgensen, G. Moruz, and T. Mølhave. Counting in the presence of memory faults. In *ISAAC*, pages 842–851, 2009.

7 V. Chen, E. Grigorescu, and R. de Wolf. Efficient and error-correcting data structures for membership and polynomial evaluation. In *STACS*, pages 203–214, 2010.

8 M. Chu, S. Kannan, and A. McGregor. Checking and spot-checking the correctness of priority queues. In *ICALP*, pages 728–739, 2007.

9 R. de Wolf. Error-correcting data structures. In *STACS*, pages 313–324, 2009.

10 I. Finocchi, F. Grandoni, and G. F. Italiano. Optimal resilient sorting and searching in the presence of memory faults. *Theor. Comput. Sci.*, 410(44):4457–4470, 2009.

11 I. Finocchi, F. Grandoni, and G. F. Italiano. Resilient dictionaries. *ACM Transactions on Algorithms*, 6(1), 2009.

12 I. Finocchi and G. F. Italiano. Sorting and searching in faulty memories. *Algorithmica*, 52(3):309–332, 2008.

13 D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, 1975.

14 B. L. Jacob, S. W. Ng, and D. T. Wang. *Memory Systems: Cache, DRAM, Disk.* Morgan Kaufmann, 2008.

15 A. G. Jørgensen, G. Moruz, and T. Mølhave. Priority queues resilient to memory faults. In *WADS*, pages 127–138, 2007.

16 R. M. Karp and M. O. Rabin. Efficient randomized pattern-matching algorithms. *IBM J. Res. Dev.*, 31(2):249–260, 1987.

17 F. T. Leighton, Y. Ma, and C. G. Plaxton. Breaking the $\theta(n \log^2 n)$ barrier for sorting with faults. *J. Comput. Syst. Sci.*, 54(2):265–304, 1997.

18 A. Pelc. Searching games with errors - fifty years of coping with liars. *Theor. Comput. Sci.*, 270(1–2):71–109, 2002.

19 M. O. Rabin. Probabilistic algorithm for testing primality. *J. of Number Th.*, 12(1):128–138, 1980.

20 B. Schroeder, E. Pinheiro, and W. D. Weber. DRAM errors in the wild: a large-scale field study. In *SIGMETRICS/Performance*, pages 193–204, 2009.

21 E. Ukkonen. Algorithms for approximate string matching. *Information and Control*, 64(1–3):100–118, 1985.

Tight bounds for rumor spreading in graphs of a given conductance*

George Giakkoupis

Université Paris 7
Paris, France
`ggiak@liafa.jussieu.fr`

Abstract

We study the connection between the rate at which a rumor spreads throughout a graph and the *conductance* of the graph—a standard measure of a graph's expansion properties. We show that for any n-node graph with conductance ϕ, the classical PUSH-PULL algorithm distributes a rumor to all nodes of the graph in $O(\phi^{-1}\log n)$ rounds with high probability (w.h.p.). This bound improves a recent result of Chierichetti, Lattanzi, and Panconesi [6], and it is tight in the sense that there exist graphs where $\Omega(\phi^{-1}\log n)$ rounds of the PUSH-PULL algorithm are required to distribute a rumor w.h.p.

We also explore the PUSH and the PULL algorithms, and derive conditions that are both necessary and sufficient for the above upper bound to hold for those algorithms as well. An interesting finding is that every graph contains a node such that the PULL algorithm takes $O(\phi^{-1}\log n)$ rounds w.h.p. to distribute a rumor started at that node. In contrast, there are graphs where the PUSH algorithm requires significantly more rounds for any start node.

1998 ACM Subject Classification G.3 [Mathematics of Computing]: Probability and Statistics

Keywords and phrases Conductance, rumor spreading

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.57

1 Introduction

Gossip-based algorithms have become a prominent paradigm for designing simple, efficient, and robust protocols for disseminating information in large networks. Perhaps the most basic and most well-studied example of a gossip-based information-dissemination algorithm is the, so-called, *rumor-spreading model.* The algorithm proceeds in a sequence of synchronous rounds. Initially, in round 0, an arbitrary *start* node receives a piece of information, called the *rumor.* This rumor is then spread iteratively to other nodes: In each round, every *informed* node (i.e., every node that received the rumor in a previous round) chooses a random neighbor to which it transmits the rumor. This is the *PUSH* version of the rumor-spreading model. The *PULL* version is symmetric: In each round, every *uninformed* node chooses a random neighbor, and if that neighbor knows the rumor it transmits it to the uniformed node. Finally, the *PUSH-PULL* algorithm is the combination of both strategies: In each round, every node chooses a random neighbor to transmit the rumor to, if the node knows the rumor, or to request the rumor from, otherwise.

The above three rumor-spreading algorithms were proposed in [8], in the context of maintaining distributed replicated database systems. Subsequently, these algorithms (and variations of them) have been used in various applications, such as failure detection [27],

[*] This work was supported by the European project EULER and the INRIA project GANG.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 57–68
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

resource discovery [21], and data aggregation [3]. Also, their performance has been studied theoretically for several classes of networks (see the Related Work Section).

In this paper, we investigate the relationship between the performance of rumor spreading in an arbitrary connected network (represented by an undirected graph), and the expansion properties of the network. More precisely, we look at the *broadcast time* of the above rumor-spreading algorithms, i.e., the number of rounds until all nodes get informed—a primary measure for rumor spreading. And we study its connection to the *conductance* of the network—one of the most studied measures of graph expansion. Roughly speaking, the conductance of a connected graph is a value ϕ in the range $0 < \phi \leq 1$, which is large for graphs that are well connected (e.g., the complete graph), and small for graphs that are not (e.g., graphs with communication bottlenecks).

A connection between broadcast time and conductance has been observed in several works, e.g., in [23, 25, 1, 10], where upper bounds on the broadcast time were obtained for various graph topologies based, essentially, on lower bounds on the conductance. In [5], Chierichetti, Lattanzi, and Panconesi posed the question whether rumor spreading is fast in *all* graphs with high conductance. For the PUSH and the PULL algorithms the answer is negative; as observed in [6], a star with n vertices has constant conductance but the expected broadcast time for a random start node is $\Omega(n)$ rounds. For the PUSH-PULL algorithm, however, the answer to the above question is positive. In [7], it was shown that for any graph and any start node, the broadcast time of the PUSH-PULL algorithm is $O(\phi^{-2} \log n)$ rounds, with high probability (*w.h.p.*).[1] It was also noted in [7] that this result suggests a justification as to why rumors spread quickly among humans, since experimental studies have shown that social networks have high conductance. The above bound was subsequently improved to $O\big((\log \phi^{-1})^2 \phi^{-1} \log n\big)$ rounds w.h.p., in [6]. Further, it was shown there that this bound is by at most a $(\log \phi^{-1})^2$-factor larger than the optimal bound. More precisely, it was shown that for any $\phi \geq 1/n^{1-\epsilon}$, there are n-node graphs with conductance at least ϕ and diameter $\Omega(\phi^{-1} \log n)$. Finally, the authors of [6] provided a sufficient condition for their upper bound to hold for the PUSH and the PULL algorithms as well. This condition states that for any edge, the ratio of the degrees of its two endpoints is bounded by a constant.

Two other important measures of a graph's expansion properties are *edge* and *vertex expansion.* The authors of [5] described a graph with constant edge expansion in which the expected broadcast time of the PUSH-PULL algorithm for a random start node is $\Omega(\sqrt{n})$. The question whether high vertex expansion yields fast rumor spreading (also posed in [5]) is largely open; in a very recent work [26], it was shown that for *regular* graphs this is true.

Our Contributions. We saw that an upper bound of $O\big((\log \phi^{-1})^2 \phi^{-1} \log n\big)$ rounds w.h.p. is known for the broadcast time of the PUSH-PULL algorithm in *any* graph; and $\Omega(\phi^{-1} \log n)$ rounds are required for *some* graph with n nodes and conductance ϕ, for any n and $\phi \geq 1/n^{1-\epsilon}$. Our first contribution is the following result, which closes the gap between these two bounds.

▶ **Theorem 1.** *For any graph on n vertices and any start vertex, the broadcast time of the PUSH-PULL algorithm is $O(\phi^{-1} \log n)$ rounds w.h.p.*

We also show that Theorem 1 is tight for $\phi = \Omega(1/n)$—not just for $\phi \geq 1/n^{1-\epsilon}$ as it was previously known. Clearly, the theorem is not tight for $\phi = o(1/n)$, since the broadcast time of the PUSH algorithm is known to be $O(n \log n)$ rounds w.h.p. for any graph [16].

[1] By "with high probability" we mean with probability $1 - O(n^{-c})$, for an arbitrary constant $c > 0$.

The proof of Theorem 1 is based on an analysis of the PUSH and the PULL algorithms. We show that in any graph, the broadcast time of the PULL algorithm is $O(\phi^{-1}\log n)$ rounds w.h.p., *if the start node has degree* Δ, the maximum degree of the graph. Also, based on the symmetry between the PULL and the PUSH algorithms, we show that for any start node, the PUSH algorithm takes $O(\phi^{-1}\log n)$ rounds w.h.p. to inform a node of degree Δ. Therefore, w.h.p. the PUSH-PULL algorithm takes $O(\phi^{-1}\log n)$ rounds to inform a node of degree Δ, and $O(\phi^{-1}\log n)$ additional rounds to inform the remaining nodes.

Our analysis is different than previous approaches. Specifically, the proof in [7] is based on a connection between rumor spreading and a spectral sparsification process; and the proof in [6] analyzes the PUSH-PULL process directly. Still, our analysis uses some ideas from [6].

Recall that high conductance does not always yield short broadcast times for the PUSH and the PULL algorithms. Our second contribution is that we derive conditions guaranteing a broadcast time of $O(\phi^{-1}\log n)$ rounds w.h.p for those algorithms. As mentioned above, in the proof of Theorem 1 we show that one such condition for the PULL algorithm is that the start node have degree Δ. We extend this result as follows. Let δ denote the minimum degree of the graph.

▶ **Theorem 2.** *(a) For any graph on* n *vertices and any start vertex with degree* $\Omega(\Delta(\phi+\delta^{-1}))$*, the broadcast time of the PULL algorithm is* $O(\phi^{-1}\log n)$ *rounds w.h.p. (b) If, in particular,* $\Delta = O(1/\phi)$ *then the above bound on the broadcast time holds for any start vertex.*

Further, we show that the conditions specified in Theorem 2 are optimal, in the sense that for any given ϕ, δ, Δ, d with $\Delta = \omega(1/\phi)$ and $d = o(\Delta(\phi+\delta^{-1}))$, there is a graph with those ϕ, δ, Δ, and with a start node of degree d such that the broadcast time of the PULL algorithm is $\omega(\phi^{-1}\log n)$ with non-negligible probability (i.e., with probability $n^{-o(1)}$).

Note that Theorem 2(a) does not hold for the PUSH algorithm: a star on n vertices has constant conductance, but the broadcast time of the PUSH algorithm is at least $n-1$.

From Theorem 2(a) it follows that if $\delta = \Omega(\Delta(\phi+\delta^{-1}))$ then the broadcast time of the PULL algorithm is $O(\phi^{-1}\log n)$ w.h.p. for *all* start nodes. This, and Theorem 2(b), are also true for the PUSH algorithm, by the symmetry argument used in the proof of Theorem 1.

Finally, we also tighten the result of [6] for the PUSH and the PULL algorithms. We show that if, for any edge, the ratio of the degrees of its endpoints is bounded, then the broadcast time of those algorithms is $O(\phi^{-1}\log n)$ rounds w.h.p., for any start node.

Related Work. The broadcast time of the PUSH algorithm has been analyzed for various graph topologies, including the complete graph [19, 24], the hypercube and random graphs [16], star and Cayley graphs [12, 13], regular graphs [15], and random regular graphs [17].

Besides the broadcast time, another performance measure of interest is the total number of transmissions of the rumor. Fewer transmissions are typically achieved using the PUSH-PULL algorithm. The broadcast time and the number of transmissions of the PUSH-PULL algorithm (and variations of it) have been analyzed for the complete graph [22], random graphs [11, 14], and random regular graphs [1]. The problem of minimizing the total communication complexity (i.e., the total number of bits transmitted) was studied in [18] for the complete graph.

A quasi-random variant of the rumor-spreading model was proposed in [9], as a means to reduce the amount of randomness. In the quasi-random model, each node has a (cyclic) list of its neighbors in which it just chooses a random starting position—instead of choosing a new random position in each round. This model was shown to be at least as efficient as the classical rumor-spreading model for several families of graphs [9, 10]. The problem of further reducing the amount of randomness was studied in [20].

The problem of rumor spreading in arbitrary graphs and its connection to the graph's expansion properties were also studied in [3, 23], in the context of gossip-based data aggregation. In both papers, the data-aggregation protocols proposed employ generalizations of the PUSH-PULL algorithm with non-uniform selection probabilities: in each round, node v chooses its neighbor u with probability $p_{v,u}$. Under certain symmetry conditions for the matrix of $p_{v,u}$, upper bounds on the broadcast time were established, as a function of certain measures of this matrix that resemble graph conductance. These results, however, are not directly comparable to our results. In particular, as observed in [6], there are graphs with high conductance for which the above approaches yield large bounds for the broadcast time.

The problem of *partial* rumor spreading, where it suffices that the rumor be spread to a constant fraction of the nodes, was studied in [4]. There, a refinement of graph conductance, called *weak conductance*, was introduced, and it was shown that high weak conductance always implies fast partial rumor spreading (using the PUSH-PULL algorithm), even if the (standard) conductance is small.

Paper organization. We begin with some definitions and notations, in Section 2. Section 3, which constitutes the largest part of the paper, contains the analysis of the PULL algorithm, including the proof of Theorem 2. In Section 4, we provide a result on the symmetry between the PUSH and PULL algorithms, which allows us to derive the properties of the PUSH algorithm from the analysis of the PULL algorithm. Finally, in Section 5, we analyze the PUSH-PULL algorithm and prove Theorem 1 using results from Sections 3 and 4.

2 Preliminaries

We consider an arbitrary connected network, represented by an undirected graph $G = (V, E)$. The degree of a vertex $v \in V$ is denoted $d(v)$. By Δ we denote the maximum degree of G, $\Delta = \max_{v \in V} d(v)$, and by δ we denote the minimum degree. The *volume* of a subset of vertices $S \subseteq V$ is the sum of the degrees of the vertices in S, $\mathrm{vol}(S) = \sum_{v \in S} d(v)$. Note that $\mathrm{vol}(V) = 2|E|$. By $\mathrm{cut}(S, V - S)$ we denote the set of edges crossing the partition $\{S, V - S\}$ of V, i.e., $\mathrm{cut}(S, V - S) = \{\{v, u\} \in E : v \in S,\ u \in V - S\}$. The *conductance* ϕ of G is defined as

$$\phi = \min_{S \subseteq V,\, \mathrm{vol}(S) \leq |E|} \frac{|\,\mathrm{cut}(S, V - S)|}{\mathrm{vol}(S)}.$$

It is easy to see that $0 < \phi \leq 1$. (It is $\phi \neq 0$ because graph G is connected.) Also,

▶ **Observation 3.** *For any $S \subseteq V$, $|\,\mathrm{cut}(S, V - S)| \geq \lceil \phi \cdot \min\{\mathrm{vol}(S), \mathrm{vol}(V - S)\} \rceil$.*

We will denote by S_i the set of informed vertices at the *end* of round i of the rumor-spreading algorithm, and by U_i the set of uninformed vertices at that time, $U_i = V - S_i$. S_0 and U_0 denote the corresponding sets initially. To simplify notation, we will assume that S_0 can be any non-empty subset of vertices—we do not require that $|S_0| = 1$.

3 PULL Algorithm

In Section 3.1, we establish a general upper bound on the broadcast time of the PULL algorithm, for any initial set of informed vertices. In Section 3.2, we build upon and refine this result to derive conditions that guarantee broadcast times of $O(\phi^{-1} \log n)$ rounds. More precisely, we prove Theorem 2 and demonstrate its optimality, and we show that a condition proposed in [6] also achieves the above broadcast time.

3.1 An Upper Bound on the Broadcast Time

The main result of this section is the following high-probability bound on the broadcast time for an arbitrary initial set of informed vertices. Recall that Δ is the maximum degree of G.

▶ **Lemma 4.** *For any initial set of informed vertices $S_0 \subseteq V$ and any fixed $\beta > 0$, all vertices get informed in at most $50(\beta+2)\log n\big(\phi^{-1} + \Delta/\lceil\phi\operatorname{vol}(S_0)\rceil\big)$ rounds of the PULL algorithm, with probability $1 - O(n^{-\beta})$.*

Note that if $\Delta/\lceil\phi\operatorname{vol}(S_0)\rceil = O(1/\phi)$ then the broadcast time is $O(\phi^{-1}\log n)$ w.h.p.

To prove Lemma 4 we divide the execution of the algorithm into three phases: The first phase lasts until the total volume of informed vertices becomes at least Δ; the second lasts until this volume exceeds $|E|$, i.e., it exceeds one half of the total volume of the graph; and the third lasts until all vertices get informed. We measure progress in the first two phases by the increase in the volume of informed vertices; and in the third phase by the decrease in the volume of uninformed vertices. For each phase, the next lemma gives upper bounds on the number of rounds until "significant" progress is made with constant probability.

▶ **Lemma 5.**

(a) *If $\operatorname{vol}(S_0) < \Delta$ then $\Pr\big(\operatorname{vol}(S_i) \geq \Delta\big) \geq 1/2$, for $i \geq 4\Delta/\lceil\phi\operatorname{vol}(S_0)\rceil$.*

(b) *If $\Delta \leq \operatorname{vol}(S_0) \leq |E|$ then $\Pr\big(\operatorname{vol}(S_i) \geq \min\{2\operatorname{vol}(S_0),\, |E|+1\}\big) \geq 1/2$, for $i \geq 4/\phi$.*

(c) *If $\operatorname{vol}(S_0) > |E|$ then $\Pr\big(\operatorname{vol}(U_i) \leq \operatorname{vol}(U_0)/2\big) \geq 1/2$, for $i \geq 6/\phi$.*

The proof of Lemma 5 proceeds as follows. Consider part (a)—for parts (b) and (c) the reasoning is similar. Consider round i. At the beginning of the round there are at least $\phi\operatorname{vol}(S_{i-1}) \geq \phi\operatorname{vol}(S_0)$ edges between informed and uninformed vertices. We fix $\lceil\phi\operatorname{vol}(S_0)\rceil$ of these edges arbitrarily before round i is executed, and then count the total volume L_i of the vertices that get informed in round i due to the rumor being transmitted *through those edges*. Clearly, L_i is a lower bound on the total volume of the vertices informed in round i. Thus, to prove (a) it suffices to show that $\sum_{k\leq i} L_k \geq \Delta - \operatorname{vol}(S_0)$ with probability at least $1/2$. By employing a martingale argument we compute the expectation and the variance of $\sum_{k\leq i} L_k$, and then we bound $\sum_{k\leq i} L_k$ using Chebyshev's inequality.

The approach used to prove Lemma 5 is at the heart of our analysis, and it is also used to prove analogous results in Section 3.2.

Proof of Lemma 5. **(a)** Let $L_1, L_2, \ldots$ be a sequence of random variables with L_i, for $i \geq 1$, be defined as follows. We distinguish two cases:

- If $\operatorname{vol}(S_{i-1}) \leq |E|$, then, by Observation 3, $|\operatorname{cut}(S_{i-1}, U_{i-1})| \geq \lceil\phi\operatorname{vol}(S_{i-1})\rceil \geq \lceil\phi\operatorname{vol}(S_0)\rceil$. Let E_i be an arbitrary subset of $\operatorname{cut}(S_{i-1}, U_{i-1})$ consisting of $M = \lceil\phi\operatorname{vol}(S_0)\rceil$ edges. Set E_i is (arbitrarily) fixed at the *beginning* of round i—before the round is executed. Then L_i is the total volume of the vertices that get informed in round i as a result of the rumor being transmitted through edges in E_i. Formally, for each vertex $u \in U_{i-1}$, let $L_{i,u}$ be the 0/1 random variable with $L_{i,u} = 1$ if and only if in round i vertex u receives the rumor through some edge in E_i. Then, $L_i = \sum_{u\in U_{i-1}} L_{i,u} d(u)$.
- If $\operatorname{vol}(S_{i-1}) > |E|$, then $L_i = M$.

We will show the following results for the expectation and the variance of the sum of L_i.

▶ **Claim 6.** $\mathbf{E}[\sum_{k\leq i} L_k] = iM$ *and* $\mathbf{Var}(\sum_{k\leq i} L_k) \leq iM\Delta$.

Using this claim, the lemma follows by Chebyshev's inequality: Let $\mu = \mathbf{E}[\sum_{k\le i} L_k] = iM$. Note that for $i \ge 4\Delta/M$, $\mu > \Delta$. So,

$$\Pr\Big(\sum_{k\le i} L_k < \Delta\Big) \le \Pr\Big(\Big|\sum_{k\le i} L_k - \mu\Big| > \mu - \Delta\Big) \le \frac{\mathbf{Var}(\sum_{k\le i} L_k)}{(\mu-\Delta)^2} \le \frac{iM\Delta}{(iM-\Delta)^2} < 1/2, \quad (1)$$

for $i \ge 4\Delta/M$. Note that if $\mathrm{vol}(S_i) < \Delta$ then $\sum_{k\le i} L_k < \Delta$, because $\sum_{k\le i} L_k$ cannot be larger than the total volume of all vertices informed since round 1 and thus $\sum_{k\le i} L_k \le \mathrm{vol}(S_i) - \mathrm{vol}(S_0) < \Delta$. Hence, $\Pr(\mathrm{vol}(S_i) < \Delta) \le \Pr(\sum_{k\le i} L_k < \Delta) < 1/2$, for $i \ge 4\Delta/M$.

To complete the proof of part (a) it remains to show Claim 6, which we do next.

Expectation of the Sum of L_i: For $i \ge 0$, define $\mathcal{L}_i = \sum_{k\le i}(L_k - M)$. Let $\mathcal{F}_i$ be the σ-algebra generated by all the choices of the algorithm in the first i rounds. It is easy to see that the sequence $\mathcal{L}_0, \mathcal{L}_1, \ldots$ is a martingale with respect to the filter $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \ldots$:

- If $\mathrm{vol}(S_{i-1}) \le |E|$,

$$\begin{aligned}\mathbf{E}[\mathcal{L}_i - \mathcal{L}_{i-1} \,|\, \mathcal{F}_{i-1}] = \mathbf{E}[L_i - M \,|\, \mathcal{F}_{i-1}] &= \mathbf{E}\Big[\sum_{u\in U_{i-1}} L_{i,u} d(u) \,\Big|\, \mathcal{F}_{i-1}\Big] - M \\ &= \sum_{u\in U_{i-1}} \mathbf{E}[L_{i,u} \,|\, \mathcal{F}_{i-1}] \cdot d(u) - M,\end{aligned}$$

where the last relation holds because U_{i-1} is $\mathcal{F}_{i-1}$-measurable. For any $u \in U_{i-1}$,

$$\mathbf{E}[L_{i,u} \,|\, \mathcal{F}_{i-1}] = \Pr(L_{i,u} = 1 \,|\, \mathcal{F}_{i-1}) = g_i(u)/d(u), \quad (2)$$

where $g_i(u)$ is the number of edges in E_i that are incident to u. Note that

$$\sum_{u\in U_{i-1}} g_i(u) = |E_i| = M, \quad (3)$$

since each edge in E_i is incident to exactly one $u \in U_{i-1}$. Combining the above yields $\mathbf{E}[\mathcal{L}_i - \mathcal{L}_{i-1} \,|\, \mathcal{F}_{i-1}] = \sum_{u\in U_{i-1}} g_i(u) - M = M - M = 0$.

- If $\mathrm{vol}(S_{i-1}) > |E|$, then $\mathcal{L}_i - \mathcal{L}_{i-1} = L_i - M = M - M = 0$.

So, in both cases, $\mathbf{E}[\mathcal{L}_i - \mathcal{L}_{i-1} \,|\, \mathcal{F}_{i-1}] = 0$, which yields $\mathbf{E}[\mathcal{L}_i] = \mathbf{E}[\mathcal{L}_0] = 0$. Substituting to this the definition of $\mathcal{L}_i$, we obtain the desired formula for the expectation, $\mathbf{E}[\sum_{k\le i} L_k] = iM$.

Variance of the Sum of L_i:

$$\begin{aligned}\mathbf{E}[\mathcal{L}_i^2 \,|\, \mathcal{F}_{i-1}] &= \mathbf{E}[((\mathcal{L}_i - \mathcal{L}_{i-1}) + \mathcal{L}_{i-1})^2 \,|\, \mathcal{F}_{i-1}] \\ &= \mathbf{E}[(\mathcal{L}_i - \mathcal{L}_{i-1})^2 \,|\, \mathcal{F}_{i-1}] + \mathcal{L}_{i-1}^2 + 2\mathbf{E}[\mathcal{L}_i - \mathcal{L}_{i-1} \,|\, \mathcal{F}_{i-1}] \cdot \mathcal{L}_{i-1} \\ &= \mathbf{E}[(L_i - M)^2 \,|\, \mathcal{F}_{i-1}] + \mathcal{L}_{i-1}^2, \quad (4)\end{aligned}$$

since $\mathbf{E}[\mathcal{L}_i - \mathcal{L}_{i-1} \,|\, \mathcal{F}_{i-1}] = 0$. We bound $\mathbf{E}[(L_i - M)^2 \,|\, \mathcal{F}_{i-1}]$ as follows:

- If $\mathrm{vol}(S_{i-1}) \le |E|$, then, by the definition of L_i and Equation (3),

$$\begin{aligned}\mathbf{E}[(L_i - M)^2 \,|\, \mathcal{F}_{i-1}] &= \mathbf{E}\Big[\Big(\sum_{u\in U_{i-1}} \big(L_{i,u} d(u) - g_i(u)\big)\Big)^2 \,\Big|\, \mathcal{F}_{i-1}\Big] \\ &= \sum_{u\in U_{i-1}} \mathbf{E}[(L_{i,u} d(u) - g_i(u))^2 \,|\, \mathcal{F}_{i-1}],\end{aligned}$$

where the last relation holds because $\mathbf{E}\big[(L_{i,u} d(u) - g_i(u))(L_{i,u'} d(u') - g_i(u')) \,\big|\, \mathcal{F}_{i-1}\big] = 0$, for any $u, u' \in U_{i-1}$ with $u \ne u'$. This last statement is true because, by (2),

$\mathbf{E}[L_{i,u}d(u) - g_i(u) \,|\, \mathcal{F}_{i-1}] = 0$, and because the random variables $L_{i,u}$, $u \in U_{i-1}$, are mutually independent conditionally on $\mathcal{F}_{i-1}$. Using again (2) and (3), we get

$$\begin{aligned}\sum_{u\in U_{i-1}} \mathbf{E}[(L_{i,u}d(u) - g_i(u))^2 \,|\, \mathcal{F}_{i-1}] &= \sum_{u\in U_{i-1}} \left(\mathbf{E}[L_{i,u}^2 (d(u))^2 \,|\, \mathcal{F}_{i-1}] - (g_i(u))^2\right) \\ &\le \sum_{u\in U_{i-1}} \mathbf{E}[L_{i,u}^2 (d(u))^2 \,|\, \mathcal{F}_{i-1}] = \sum_{u\in U_{i-1}} \mathbf{E}[L_{i,u} \,|\, \mathcal{F}_{i-1}] \cdot (d(u))^2 \\ &= \sum_{u\in U_{i-1}} g_i(u)d(u) \le \sum_{u\in U_{i-1}} g_i(u)\Delta = M\Delta.\end{aligned}$$

Therefore, $\mathbf{E}[(L_i - M)^2 \,|\, \mathcal{F}_{i-1}] \le M\Delta$.

- If $\mathrm{vol}(S_{i-1}) > |E|$, the last inequality is still true, since $L_i = M$.

By applying the above to (4) yields $\mathbf{E}[\mathcal{L}_i^2 \,|\, \mathcal{F}_{i-1}] \le M\Delta + \mathcal{L}_{i-1}^2$, and recursively we obtain $\mathbf{E}[\mathcal{L}_i^2] \le iM\Delta$. The desired bound for $\mathbf{Var}(\sum_{k\le i} L_k)$ then follows by observing that $\mathbf{Var}(\sum_{k\le i} L_k) = \mathbf{E}[\mathcal{L}_i^2]$. This completes the proof of Claim 6, and of Lemma 5(a).

(b) We consider the same sequence of random variables $L_1, L_2, \dots$ as in part (a). Similarly to (1), by using Claim 6 and Chebyshev's inequality we obtain that

$$\Pr\Big(\sum_{k\le i} L_k < \mathrm{vol}(S_0)\Big) \le \frac{iM\Delta}{(iM - \mathrm{vol}(S_0))^2} \le \frac{iM\,\mathrm{vol}(S_0)}{(iM - \mathrm{vol}(S_0))^2} < 1/2,$$

for $i \ge 4\,\mathrm{vol}(S_0)/M$, and thus, for $i \ge 4/\phi$. Part (b) then follows by observing that if $\mathrm{vol}(S_i) < \min\{2\,\mathrm{vol}(S_0), |E|+1\}$ then $\sum_{k\le i} L_k \le \mathrm{vol}(S_i) - \mathrm{vol}(S_0) < \mathrm{vol}(S_0)$, and thus, $\Pr\big(\mathrm{vol}(S_i) < \min\{2\,\mathrm{vol}(S_0), |E|+1\}\big) \le \Pr\big(\sum_{k\le i} L_k < \mathrm{vol}(S_0)\big)$.

(c) Unlike in parts (a) and (b), the set of uninformed vertices has now a smaller volume than the set of informed vertices. So, by Observation 3, $|\,\mathrm{cut}(S_i, U_i)| \ge \lceil \phi\,\mathrm{vol}(U_i)\rceil$. We consider the sequence $L_1, L_2, \dots$ of random variables, with L_i defined as follows:

- If $\mathrm{vol}(U_{i-1}) > \mathrm{vol}(U_0)/2$, we let E_i be an arbitrary subset of $\mathrm{cut}(S_{i-1}, U_{i-1})$ consisting of $M = \lceil \phi\,\mathrm{vol}(U_0)/2\rceil$ edges. (E_i is fixed at the beginning of round i.) As before, L_i is the total volume of the vertices that get informed in round i as a result of the rumor being transmitted through edges in E_i.
- If $\mathrm{vol}(U_{i-1}) \le \mathrm{vol}(U_0)/2$, then $L_i = M$.

Similarly to Claim 6, we can show that $\mathbf{E}[\sum_{k\le i} L_k] = iM$ and $\mathbf{Var}(\sum_{k\le i} L_k) \le iM\,\mathrm{vol}(U_0)$. For the latter we use the fact that the degree of any uninformed vertex is at most $\mathrm{vol}(U_0)$. As before, by Chebyshev's inequality, we can show that $\Pr\big(\sum_{k\le i} L_k < \mathrm{vol}(U_0)/2\big) < 1/2$, for $i \ge 6/\phi$. Part (c) then follows by observing that if $\mathrm{vol}(U_i) > \mathrm{vol}(U_0)/2$ then $\sum_{k\le i} L_k \le \mathrm{vol}(U_0) - \mathrm{vol}(U_i) < \mathrm{vol}(U_0)/2$. ◀

Using the bounds of Lemma 5, Lemma 4 follows easily:

Proof of Lemma 4. By Lemma 5(a), if $\mathrm{vol}(S_i) < \Delta$ then, with probability 1/2, it takes at most $\lceil 4\Delta/\lceil\phi\,\mathrm{vol}(S_i)\rceil\rceil \le 5\Delta/\lceil\phi\,\mathrm{vol}(S_i)\rceil$ additional rounds until the total volume of informed vertices becomes at least Δ. Thus, if $\mathrm{vol}(S_0) < \Delta$, the probability that $\mathrm{vol}(S_t) < \Delta$ for $t = 2\beta\ln n \cdot (5\Delta/\lceil\phi\,\mathrm{vol}(S_0)\rceil)$ is at most $(1-1/2)^{2\beta\ln n} \le e^{-2\beta\ln n/2} = n^{-\beta}$.

By Lemma 5(b), if $\Delta \le \mathrm{vol}(S_i) \le |E|$ then, with probability 1/2, it takes at most $\lceil 4/\phi\rceil$ rounds until the total volume of informed vertices is increased to at least $\min\{2\,\mathrm{vol}(S_i), |E|+1\}$. Now, divide the execution of the algorithm into phases of $\lceil 4/\phi\rceil$ rounds each, starting from the end of the first round i with $\mathrm{vol}(S_i) \ge \Delta$. A phase is *successful* if the total volume of informed vertices at the end of the phase is at least $\min\{2\operatorname{vol}(S), |E|+1\}$, where S is the set of informed vertices at the beginning of the phase. (Note that if $\operatorname{vol}(S) \geq |E|+1$ then the phase is always successful.) Then, for any k, the probability that the k-th phase is successful is at least $1/2$, regardless of the outcome of the previous $k-1$ phases. From this (and a simple coupling argument), the number of successful phases among the first k phases is (stochastically) greater or equal to the binomial random variable $B(k, 1/2)$. So, by Chernoff bounds, the probability that fewer than $m = \log|E|$ of the first $k = (2\beta+4)m$ phases are successful is at most equal to

$$\Pr(B(k,1/2) < m) = \Pr(k/2 - B(k,1/2) > k/2 - m) \leq e^{-2(k/2-m)^2/k} \leq e^{-\beta m} = O(n^{-\beta}),$$

since $|E| \geq n-1$. And since at most m successful phases are required until the total volume of informed vertices exceeds $|E|$, it follows that with probability $1 - O(n^{-\beta})$ the number of rounds required is at most $k\lceil 4/\phi\rceil = (2\beta+4)\log(|E|)\lceil 4/\phi\rceil \leq (2\beta+4)(2\log n)(5/\phi)$.

Finally, by Lemma 5(c), if $\operatorname{vol}(S_i) > |E|$ then, with probability $1/2$, it takes at most $\lceil 6/\phi\rceil$ rounds until the total volume of uninformed vertices is halved. By similar reasoning as before, we can show that once the volume of informed vertices has exceeded $|E|$, then $(2\beta+4)(2\log n)(7/\phi)$ rounds suffice to inform all nodes with probability $1 - O(n^{\beta})$.

Combing all the above and applying the union bound, we obtain that with probability $1 - O(n^{\beta})$ all vertices get informed within $50(\beta+2)\log n\big(\phi^{-1} + \Delta/\lceil\phi\operatorname{vol}(S_0)\rceil\big)$ rounds. ◀

3.2 Conditions for Rumor Spreading in $O(\phi^{-1}\log n)$ Rounds

3.2.1 Derivation of Theorem 2

Lemma 4 implies that if $\Delta/\lceil\phi\operatorname{vol}(S_0)\rceil = O(1/\phi)$, the broadcast time is $O(\phi^{-1}\log n)$ rounds w.h.p. Theorem 2(b) follows then directly, since $\Delta/\lceil\phi\operatorname{vol}(S_0)\rceil \leq \Delta$, for any S_0. Also, the following weaker version of Theorem 2(a) is immediate, because if the degree of the start vertex is $\Omega(\Delta)$ then $\operatorname{vol}(S_0) = \Omega(\Delta)$ and $\Delta/\lceil\phi\operatorname{vol}(S_0)\rceil \leq \Delta/\phi\operatorname{vol}(S_0) = O(1/\phi)$.

▶ **Corollary 7.** *For any start vertex of degree $\Omega(\Delta)$, the broadcast time of the PULL algorithm is $O(\phi^{-1}\log n)$ rounds w.h.p.*

This result is weaker than Theorem 2(a) because $\phi + \delta^{-1} = O(1)$. However, it will suffice for the purposes of proving Theorem 1 (in Section 5).

Next we describe the proof of Theorem 2(a). Recall that Lemma 4, on which the proof of Corollary 7 was based, assumes that S_0 may be any subset of vertices. Under this assumption, the size of $\operatorname{cut}(S_0, U_0)$ can be as small as $\lceil\phi\operatorname{vol}(S_0)\rceil$. However, if S_0 consists of a single vertex, then $|\operatorname{cut}(S_0, U_0)| = \operatorname{vol}(S_0)$, which can be significantly larger than $\lceil\phi\operatorname{vol}(S_0)\rceil$. This observation is a key ingredient in our proof.

We begin by observing that if S_0 consists of a single vertex, then the size of $\operatorname{cut}(S_i, U_i)$ remains $\Omega(\operatorname{vol}(S_0))$ until $\operatorname{vol}(S_i)$ increases to at least $\Omega(\delta\operatorname{vol}(S_0))$. More precisely, suppose that $S_0 = \{v\}$; so, $\operatorname{vol}(S_0) = |\operatorname{cut}(S_0, U_0)| = d(v)$. Then, $|\operatorname{cut}(S_i, U_i)| \geq \operatorname{vol}(S_0) - |S_i| + 1$, because all the $\operatorname{vol}(S_0)$ edges of the start vertex v are initially incident to uniformed vertices; and each new vertex that gets informed is incident to at most one of those edges. Also, clearly, $\operatorname{vol}(S_i) \geq |S_i| \cdot \delta$, thus, $|S_i| \leq \operatorname{vol}(S_i)/\delta$. Therefore, $|\operatorname{cut}(S_i, U_i)| \geq \operatorname{vol}(S_0) - \operatorname{vol}(S_i)/\delta$. So,

▶ **Observation 8.** *If $|S_0| = 1$ and $\operatorname{vol}(S_i) \leq \delta\operatorname{vol}(S_0)/2$ then $|\operatorname{cut}(S_i, U_i)| \geq \operatorname{vol}(S_0)/2$.*

We use this result in the proof of the next lemma, which is similar to Lemma 5(a).

▶ **Lemma 9.** *Let $D = \min\{\Delta, \delta\operatorname{vol}(S_0)/2\}$. If $|S_0| = 1$ then $\Pr\big(\operatorname{vol}(S_{j+i}) \geq D \,\big|\, S_j\big) \geq 1/2$, for $i \geq 8\Delta/\operatorname{vol}(S_0)$.*

Proof. Fix the set S_j arbitrarily. As in the proof of Lemma 5(a), we consider a sequence $L_1, L_2, \dots$ of random variables, where L_i is as follows:

- If $\mathrm{vol}(S_{j+i-1}) \le D$, let E_i be an arbitrary subset of $\mathrm{cut}(S_{j+i-1}, U_{j+i-1})$ of size $M = \lceil \mathrm{vol}(S_0)/2 \rceil$, fixed before round $j+i$. (By Observation 8, $|\mathrm{cut}(S_{j+i-1}, U_{j+i-1})| \ge M$.) Then L_i is the total volume of the vertices informed in round $j+i$ through edges in E_i.
- If $\mathrm{vol}(S_{j+i-1}) > D$, then $L_i = M$.

Similarly to Claim 6, $\mathbf{E}[\sum_{k\le i} L_k] = iM$ and $\mathbf{Var}(\sum_{k\le i} L_k) \le iM\Delta$. And, similarly to (1), $\Pr\big(\sum_{k\le i} L_k < D\big) \le iM\Delta/(iM-D)^2 < 1/2$, for $i \ge 2(\Delta + D)/M$. Since $\mathrm{vol}(S_{j+i}) < D$ implies $\sum_{k\le i} L_k < D$, and since $2(\Delta+D)/M \le 8\Delta/\mathrm{vol}(S_0)$, the lemma follows. ◀

We can now derive Theorem 2(a) similarly to Lemma 4.

Proof of Theorem 2(a). Let $d = \mathrm{vol}(S_0)$ be the degree of the start vertex. By Lemma 9, the probability that the total volume of informed vertices is smaller than $D = \min\{\Delta,\, \delta d/2\}$ after $c \ln n \lceil 8\Delta/d \rceil$ rounds is at most $(1-1/2)^{c\ln n} \le n^{-c/2}$. The above number of rounds is $O(\phi^{-1}\ln n)$, since $d = \Omega(\Delta(\phi+\delta^{-1})) = \Omega(\phi\Delta)$. Thus, w.h.p., it takes $O(\phi^{-1}\ln n)$ rounds until the total volume of informed vertices becomes at least D.

Since $d = \Omega(\Delta(\phi+\delta^{-1})) = \Omega(\Delta/\delta)$, we have $D = \Omega(\Delta)$. Thus, by Lemma 4, once the total volume of informed vertices is at least D, it takes $O\big(\log n(\phi^{-1} + \Delta/\lceil \phi D\rceil)\big) = O(\phi^{-1}\ln n)$ additional rounds until all vertices get informed w.h.p. ◀

The following direct corollary of Theorem 2(a) gives a condition for rumor spreading in $O(\phi^{-1}\log n)$ rounds for *any* start vertex.

▶ **Corollary 10.** *If $\delta = \Omega(\Delta(\phi+\delta^{-1}))$, or, equivalently, $\delta = \Omega(\phi\Delta + \sqrt{\Delta})$ then, for any start vertex, the broadcast time of the PULL algorithm is $O(\phi^{-1}\log n)$ rounds w.h.p.*

3.2.2 Optimality of Theorem 2

The conditions described in Theorem 2, that the degree of the start vertex be $d = \Omega(\Delta(\phi + \delta^{-1}))$ or the maximum degree be $\Delta = O(1/\phi)$, are optimal in the following sense.

▶ **Theorem 11.** *For any ϕ, δ, Δ, d with $\delta \le d = o(\Delta(\phi+\delta^{-1}))$ and $\Delta = \omega(1/\phi)$, there exists an infinite sequence of graphs $G_1, G_2, \dots$ such that G_n has $\Theta(n)$ vertices, conductance $\Theta(\phi)$, and maximum (minimum) degree $\Theta(\Delta)$ ($\Theta(\delta)$), and it contains a start vertex of degree $\Theta(d)$ such that $\omega(\phi^{-1}\log n)$ rounds of the PULL algorithm are required to inform all vertices w.h.p.*

Proof. First we consider the case of $d = o(\phi\Delta)$. Construct the following graph: Take a Δ-regular graph R_Δ on n vertices with edge expansion $\xi = \Theta(\Delta)$. Such a graph exists since the edge expansion of a *random* Δ-regular graph is $\Theta(\Delta)$ w.h.p. [2]. The conductance of R_Δ is obviously $\xi/\Delta = \Theta(1)$. Add a vertex s of degree d and a vertex $v_{\min}$ of degree δ, choosing their neighbors arbitrarily among the vertices of R_Δ. Vertex s will be the start vertex, while $v_{\min}$ is added just to have minimum degree δ. Next we add a component to achieve conductance ϕ: Take the complete graph on Δ vertices K_Δ. Let A be an arbitrary subset of the vertices of R_Δ of size $|A| = \lfloor \phi\Delta \rfloor$. (It is $|A| > 0$ since $1 \le d = o(\phi\Delta)$). Draw edges between each vertex of K_Δ and each vertex in A. It is not hard to see that the resulting graph has the desired number of vertices, maximum and minimum degrees, and conductance. Also, since $d = o(\phi\Delta)$, the probability that no neighbor of s receives the rumor from s in $k = \lfloor \phi^{-1}\ln n \cdot (2/3)\sqrt{\phi\Delta/d} \rfloor = \omega(\phi^{-1}\ln n)$ rounds is at least

$$(1-1/\Delta)^{kd} \ge e^{-3kd/2\Delta} \ge e^{-\ln n\sqrt{d/\phi\Delta}} = n^{-o(1)},$$

where for the first inequality we used the fact that $1-x \geq e^{-3x/2}$, for $0 \leq x \leq 1/2$. So, with probability $n^{-o(1)}$, no vertex learns a rumor started at s in $O(\phi^{-1}\ln n)$ rounds.

Next we consider the complementary case, $d = \Omega(\phi\Delta)$. Since $d = o(\Delta(\phi+\delta^{-1}))$, we have $\phi = o(1/\delta)$ and $d = o(\Delta/\delta)$. Consider the following graph: Take the graph we constructed before and remove vertex s together with its incident edges. Take also $\lceil d/\delta \rceil$ copies of K_δ. Add a vertex s' of degree $\Theta(d)$ with neighbors the vertices of the $\lceil d/\delta \rceil$ δ-cliques, plus the elements of an arbitrary subset B of the vertices of R_Δ, with $|B| = \lceil \phi d\delta \rceil$. (It is $|B| = O(d)$ since $\phi = o(1/\delta)$ as we saw above.) It is not hard to see that the resulting graph has the desired number of vertices, maximum and minimum degrees, and conductance. Also, with probability $n^{-o(1)}$, no vertex in B learns a rumor started at s' in $O(\phi^{-1}\ln n)$ rounds: Since $d = o(\Delta/\delta)$ and $\Delta = \omega(1/\phi)$, we have $|B| \leq \phi d\delta + 1 = o(\phi\Delta) + 1 = o(\phi\Delta)$. Thus, the probability that no neighbor of s' in B receives the rumor from s' in $k = \lfloor \phi^{-1}\ln n \cdot (2/3)\sqrt{\phi\Delta/|B|} \rfloor = \omega(\phi^{-1}\ln n)$ rounds is at least $(1-1/\Delta)^{k|B|} \geq e^{-3k|B|/2\Delta} \geq e^{-\ln n\sqrt{|B|/\phi\Delta}} = n^{-o(1)}$. ◀

3.2.3 Bounded Ratio of the Degrees of Adjacent Vertices

It was shown in [6] that if the ratio of the degrees of any two adjacent vertices is bounded by a constant, then the broadcast time of the PULL algorithm is $O\big((\log\phi^{-1})^2\phi^{-1}\log n\big)$ rounds w.h.p., for any start vertex. By similar reasoning as in the proofs of Lemma 4 and Theorem 2(a), we can show that, in fact, the above condition yields a broadcast time of $O(\phi^{-1}\log n)$ rounds. The proof is omitted due to space limitations.

▶ **Theorem 12.** *If, for every edge $\{v,u\}$, $d(v)/d(u) = \Theta(1)$ then, the broadcast time of the PULL algorithm is $O(\phi^{-1}\log n)$ rounds w.h.p., for any start vertex.*

4 PUSH Algorithm

The analysis of the PUSH algorithm can be reduced to that of the PULL algorithm, by exploiting a symmetry between the two algorithms, described in the following result. This result is similar to Lemma 3 in [6]. Its proof is omitted due to space limitations.

▶ **Lemma 13.** *Let $\mathcal{E}_{\text{PUSH}}(v,u,t)$ denote the event that the PUSH algorithm spreads to vertex u a rumor started at vertex v in at most t rounds; and let $\mathcal{E}_{\text{PULL}}(v,u,t)$ be defined similarly. Then, $\Pr(\mathcal{E}_{\text{PUSH}}(v,u,t)) = \Pr(\mathcal{E}_{\text{PULL}}(u,v,t))$.*

Suppose that for any vertex u, the PULL algorithm distributes a rumor started at u to all vertices in at most t rounds with probability at least $1-q$. Then, by Lemma 13, for any vertex v, the PUSH algorithm spreads to a given u a rumor started at v in at most t rounds with probability at least $1-q$; and, by the union bound, if $q \leq 1/(n-1)$, the rumor started at v is spread to *all* vertices in at most t rounds with probability at least $1-(n-1)q$. Thus, if the broadcast time of the PULL algorithm is $O(\phi^{-1}\log n)$ rounds w.h.p. for any start vertex, then the same is true for the PUSH algorithm, as well. Hence, the conditions described in Section 3 guaranteing a broadcast time of $O(\phi^{-1}\log n)$ rounds w.h.p. for any start vertex, apply to the PUSH algorithm as well; specifically, Theorem 2(b), Corollary 10, and Theorem 12. Finally, Theorem 11 is also true for the PUSH algorithm for $d = \delta$. (For, otherwise, by the same reasoning as above, with the roles of the PUSH and the PULL algorithms switched, we would contradict Theorem 11.)

5 PUSH-PULL Algorithm

We prove Theorem 1, which gives a bound of $O(\phi^{-1}\log n)$ rounds w.h.p. on the broadcast time of the PUSH-PULL algorithm, and argue that this bound is tight.

Proof of Theorem 1. Fix a vertex v and let $v_{\max}$ be a vertex of maximum degree. By Corollary 7, we have that: (A) The PULL algorithm distributes a rumor from $v_{\max}$ to all other vertices (and thus to v) in $O(\phi^{-1}\log n)$ rounds w.h.p. Combining this with Lemma 13 yields: (B) The PUSH algorithm spreads to $v_{\max}$ a rumor started at v in $O(\phi^{-1}\log n)$ rounds w.h.p. The theorem now follows easily: Statement (B) implies (a fortiori) that the PUSH-PULL algorithm spreads to $v_{\max}$ a rumor started at v in $O(\phi^{-1}\log n)$ rounds w.h.p.; and, once $v_{\max}$ is informed, Statement (A) implies that from $v_{\max}$ the PUSH-PULL algorithm spreads the rumor to all vertices in $O(\phi^{-1}\log n)$ additional rounds w.h.p. ◀

The following result was shown in [6].

▶ **Lemma 14.** *For any $\phi \geq 1/n^{1-\epsilon}$, for a fixed $\epsilon > 0$, there exists an infinite sequence of graphs $G_1, G_2, \ldots$ such that G_n has $\Theta(n)$ vertices, conductance $\Theta(\phi)$, and diameter $\Omega(\phi^{-1}\log n)$.*

From this, it is immediate that rumor spreading requires $\Omega(\phi^{-1}\log n)$ rounds, if $\phi \geq 1/n^{1-\epsilon}$. Thus, the bound of Theorem 1 is asymptotically tight for $\phi \geq 1/n^{1-\epsilon}$. The next result shows this is in fact true for all $\phi = \Omega(1/n)$.

▶ **Lemma 15.** *For any ϕ with $2/(n+2) \leq \phi \leq 1/2$, there exists an infinite sequence of graphs $G_1, G_2, \ldots$ such that G_n has n vertices and conductance $\Theta(\phi)$, and, for any start vertex, $\Omega(\phi^{-1}\log n)$ rounds of the PUSH-PULL algorithm are required to inform all vertices w.h.p.*

Proof. Consider the n-vertex graph obtained by taking two stars, one with $\lceil \phi^{-1} \rceil$ vertices and another with $n - \lceil \phi^{-1} \rceil$ vertices, and connecting their centers with an edge. It is easy to see that the resulting graph has conductance $\Theta(\phi)$. We now show that for any start vertex and any constant $c > 0$, at least $c \ln n/3\phi$ rounds are required to inform all vertices with probability $1 - n^{-c}$. Let v and v' be the centers of the two stars, where v is the center of the star containing the start vertex. Let j be the round when v gets informed. (If v is the start vertex then $j = 0$.) The probability that v' is not informed by the end of round $j + i$, which happens if the rumor is not transmitted from v to v' via a PUSH or PULL operation in any of the rounds $j + 1, \ldots, j + i$, is clearly

$$\left(1 - 1/\lceil \phi^{-1} \rceil\right)^i \left(1 - 1/(n - \lceil \phi^{-1} \rceil)\right)^i \geq \left(1 - 1/\lceil \phi^{-1} \rceil\right)^{2i} \geq (1 - \phi)^{2i} \geq e^{-3i\phi},$$

where for the first inequality we used the fact that $\phi \geq 2/(n + 2)$, that for the last the fact that $1 - x \geq e^{-3x/2}$, for $0 \leq x \leq 1/2$. For $i < c \ln n/3\phi$, it is $e^{-3i\phi} > n^{-c}$. Thus, at least $c \ln n/3\phi$ rounds are required to inform all vertices with probability $1 - n^{-c}$. ◀

References

1 P. Berenbrink, R. Elsässer, and T. Friedetzky. Efficient randomised broadcasting in random regular networks with applications in peer-to-peer systems. In *Proc. 27th ACM Symp. on Principles of Distributed Computing (PODC)*, pp. 155–164, 2008.

2 B. Bollobás. The isoperimetric number of random regular graphs. *Eur. J. Comb.*, 9(3):241–244, 1988.

3 S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Gossip algorithms: design, analysis and applications. In *Proc. 24th IEEE INFOCOM*, pp. 1653–1664, 2005.

4 K. Censor-Hillel and H. Shachnai. Partial information spreading with application to distributed maximum coverage. In *Proc. 29th ACM Symp. on Principles of Distributed Computing (PODC)*, pp. 161–170, 2010.

5 F. Chierichetti, S. Lattanzi, and A. Panconesi. Rumor spreading in social networks. In *Proc. 36th Intl. Colloq. on Automata, Languages and Programming (ICALP)*, pp. 375–386, 2009.

6 F. Chierichetti, S. Lattanzi, and A. Panconesi. Almost tight bounds for rumour spreading with conductance. In *Proc. 42nd ACM Symp. on Theory of Computing (STOC)*, pp. 399–408, 2010.

7 F. Chierichetti, S. Lattanzi, and A. Panconesi. Rumour spreading and graph conductance. In *Proc. 21st ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pp. 1657–1663, 2010.

8 A. Demers, D. Greene, C. Hauser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry. Epidemic algorithms for replicated database maintenance. In *Proc. 6th ACM Symp. on Principles of Distributed Computing (PODC)*, pp. 1–12, 1987.

9 B. Doerr, T. Friedrich, and T. Sauerwald. Quasirandom rumor spreading. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pp. 773–781, 2008.

10 B. Doerr, T. Friedrich, and T. Sauerwald. Quasirandom rumor spreading: Expanders, push vs. pull, and robustness. In *Proc. 36th Intl. Colloq. on Automata, Languages and Programming (ICALP)*, pp. 366–377, 2009.

11 R. Elsässer. On the communication complexity of randomized broadcasting in random-like graphs. In *Proc. 18th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pp. 148–157, 2006.

12 R. Elsässer, U. Lorenz, and T. Sauerwald. On randomized broadcasting in star graphs. *Discrete Appl. Math.*, 157(1):126–139, 2009.

13 R. Elsässer and T. Sauerwald. Broadcasting vs. mixing and information dissemination on Cayley graphs. In *Proc. 24th Symp. on Theoretical Aspects of Computer Science (STACS)*, pp. 163–174, 2007.

14 R. Elsässer and T. Sauerwald. The power of memory in randomized broadcasting. In *Proc. 19th ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pp. 218–227, 2008.

15 R. Elsässer and T. Sauerwald. On the runtime and robustness of randomized broadcasting. *Theoretical Computer Science*, 410(36):3414–3427, 2009.

16 U. Feige, D. Peleg, P. Raghavan, and E. Upfal. Randomized broadcast in networks. *Random Structures and Algorithms*, 1(4):447–460, 1990.

17 N. Fountoulakis and K. Panagiotou. Rumor spreading on random regular graphs and expanders. In *Proc. 14th Intl. Workshop on Randomization and Computation (RANDOM)*, pp. 560–573, 2010.

18 P. Fraigniaud and G. Giakkoupis. On the bit communication complexity of randomized rumor spreading. In *Proc. 22nd ACM Symp. on Parallel Algorithms and Architectures (SPAA)*, pp. 134–143, 2010.

19 A. Frieze and G. Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Appl. Math.*, 10:57–77, 1985.

20 G. Giakkoupis and P. Woelfel. On the randomness requirements of rumor spreading. To appear in *Proc. 22nd ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2011.

21 M. Harchol-Balter, F. T. Leighton, and D. Lewin. Resource discovery in distributed networks. In *Proc. 18th ACM Symp. on Principles of Distributed Computing (PODC)*, pp. 229–237, 1999.

22 R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *Proc. 41st IEEE Symp. on Foundations of Computer Science (FOCS)*, pp. 565–574, 2000.

23 D. Mosk-Aoyama and D. Shah. Computing separable functions via gossip. In *Proc. 25th ACM Symp. on Principles of Distributed Computing (PODC)*, pp. 113–122, 2006.

24 B. Pittel. On spreading a rumor. *SIAM J. Appl. Math.*, 47(1):213–223, 1987.

25 T. Sauerwald. On mixing and edge expansion properties in randomized broadcasting. In *Proc. 18th Intl. Symp. on Algorithms and Computation (ISAAC)*, pp. 196–207, 2007.

26 T. Sauerwald and A. Stauffer. Rumor spreading and vertex expansion on regular graphs. To appear in *Proc. 22nd ACM-SIAM Symp. on Discrete Algorithms (SODA)*, 2011.

27 R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proc. 15th IFIP Intl. Conf. on Distributed Systems Platforms (Middleware)*, pp. 55–70, 1998.

Tight Bounds For Distributed MST Verification[*] [†]

Liah Kor[1], Amos Korman[2], and David Peleg[1]

1 **Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, Rehovot, 76100 Israel.**
{liah.kor,david.peleg}@weizmann.ac.il

2 **CNRS and LIAFA, Univ. Paris 7, Paris, France.**
amos.korman@liafa.jussieu.fr

Abstract

This paper establishes tight bounds for the Minimum-weight Spanning Tree (MST) verification problem in the distributed setting. Specifically, we provide an MST verification algorithm that achieves *simultaneously* $\tilde{O}(|E|)$ messages and $\tilde{O}(\sqrt{n}+D)$ time, where $|E|$ is the number of edges in the given graph G and D is G's diameter. On the negative side, we show that any MST verification algorithm must send $\Omega(|E|)$ messages and incur $\tilde{\Omega}(\sqrt{n}+D)$ time in worst case.

Our upper bound result appears to indicate that the verification of an MST may be easier than its construction, since for MST construction, both lower bounds of $\Omega(|E|)$ messages and $\Omega(\sqrt{n}+D)$ time hold, but at the moment there is no known distributed algorithm that constructs an MST and achieves *simultaneously* $\tilde{O}(|E|)$ messages and $\tilde{O}(\sqrt{n}+D)$ time. Specifically, the best known time-optimal algorithm (using $\tilde{O}(\sqrt{n}+D)$ time) requires $O(|E|+n^{3/2})$ messages, and the best known message-optimal algorithm (using $\tilde{O}(|E|)$ messages) requires $O(n)$ time. On the other hand, our lower bound results indicate that the verification of an MST is not significantly easier than its construction.

1998 ACM Subject Classification C.2.2 [Network Protocols]: Protocol verification; C.2.4 [Distributed Systems]; G.2.2[Discrete Mathematics]: Graph Theory: Graph algorithms, Graph labeling, Trees; B.8.1[Performance and Reliability]: Reliability, Testing, and Fault-Tolerance;

Keywords and phrases Distributed algorithms, distributed verification, labeling schemes, minimum-weight spanning tree

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.69

1 Introduction

1.1 Background and Motivation

The problem of efficiently computing a Minimum-weight Spanning Tree (MST) of a given weighted graph has been studied extensively in the centralized, parallel and distributed settings. Reviews on the problem in the centralized setting can be found, e.g., in the survey paper by Graham and Hell [17] or in the book by Tarjan [32] (Chapter 6). The fastest known algorithm for finding an MST is that of Pettie and Ramachandran [29], which runs in $O(|E| \cdot \alpha(|E|, n))$ time, where α is the inverse Ackermann function, n is the number of

[*] Liah Kor and David Peleg are supported by a grant from the United States-Israel Binational Science Foundation (BSF).

[†] Amos Korman is supported in part by the ANR projects ALADDIN and PROSE, and by the INRIA project GANG.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 69–80
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

vertices and $|E|$ is the number of edges in the graph. Unfortunately, a linear (in the number of edges) time algorithm for computing an MST is known only in certain cases, or by using randomization [14, 19].

The separation between computation and verification, and specifically, the question of whether verification is easier than computation, is a central issue of profound impact on the theory of computer science. In the context of MST, the verification problem (introduced by Tarjan [31]) is the following: given a weighted graph, together with a subgraph, it is required to decide whether this subgraph forms an MST of the graph. At the time it was published, the running time of the MST verification algorithm of [31] was indeed superior to the best known bound on the computational problem. Improved verification algorithms in different centralized models were then given by Harel [18], Komlòs [23], and Dixon, Rauch, and Tarjan [9], and parallel algorithms were presented by Dixon and Tarjan [10] and by King, Poon, Ramachandran, and Sinha [22]. Though it is not known whether there exists a deterministic algorithm that computes an MST in $O(|E|)$ time, the verification algorithm of [9] is in fact linear, i.e., runs in time $O(|E|)$ (the same result with a simpler algorithm was later presented by King [21] and by Buchsbaum [5]). For the centralized setting, this may indicate that the verification of an MST is indeed easier than its computation.

The problem of computing an MST received considerable attention in the distributed setting as well. Constructing such a tree distributively requires a collaborative computational effort by all the network vertices, and involves sending messages to remote vertices and waiting for their replies. The main measures considered for evaluating a distributed MST protocol are the *message* complexity, namely, the maximum number of messages sent in the worst case scenario, and the *time* complexity, namely, the maximum number of communication rounds required for the protocol's execution in the worst case scenario. The line of research on the distributed MST computation problem was initiated by the seminal work of Gallager, Humblet, and Spira [15] and culminated in the $O(n)$ time and $O(|E| + n \log n)$ messages algorithm by Awerbuch [2]. As pointed out in [2], the results of [4, 6, 13] establish an $\Omega(|E| + n \log n)$ lower bound on the number of messages required to construct a MST. Thus, the algorithm of [2] is essentially optimal.

This was the state of affairs until the mid-nineties when Garay, Kutten, and Peleg [16] initiated the analysis of the time complexity of MST construction as a function of additional parameters (other than n), and gave the first sublinear time distributed algorithm for the MST problem, running in time $O(D + n^{0.614})$, where D is the diameter of the network. This result was later improved to $O(D + \sqrt{n} \log^* n)$ by Kutten and Peleg [26]. The tightness of this latter bound was shown by Peleg and Rubinovich [28] who proved that $\tilde{\Omega}(\sqrt{n})$ is essentially[1] a lower bound on the time for constructing MST on graphs with diameter $\Omega(\log n)$. This result was complemented by the work of Lotker, Patt-Shamir and Peleg [27] that showed an $\tilde{\Omega}(\sqrt[3]{n})$ lower bound on the time required for MST construction on graphs with small diameter. Note, however, that the time-efficient algorithms of [16, 26] are not message-optimal, i.e., they take asymptotically much more than $O(|E| + n \log n)$ messages. For example, the time-optimal protocol of [26] requires sending $O(|E| + n^{3/2})$ messages. The question of whether there exists an optimal distributed algorithm for MST construction that achieves *simultaneously* $\tilde{O}(|E|)$ messages and $\tilde{O}(\sqrt{n} + D)$ time remains open.

This paper addresses the MST verification problem in the distributed setting. Here, a subgraph is given in a distributed manner, namely, some of the edges incident to every vertex are locally marked, and the collection of marked edges at all the vertices defines the

[1] $\tilde{\Omega}$ (respectively, $\tilde{O}$) is a relaxed variant of the Ω (rep., O) notation that ignores polylog factors.

subgraph; see, e.g., [7, 15, 24, 25]. The verification task requires checking distributively whether the marked subgraph is indeed an MST of the given graph. Similarly to the centralized setting, one of our major motivations is to investigate the relationships between the MST construction problem and the related verification problem. Moreover, from an applicative point of view, since faults are much more likely to occur in the distributed setting, the verification task in the distributed setting is more practically significant than in the centralized setting. Finally, we note that generally, investigating the "simpler" verification problem may lead to breakthroughs in the efforts for solving the corresponding construction problem. This was indeed useful in the centralized setting, where the verification algorithm of [9], that determines for each edge whether it "improves" a given MST candidate, was used as a subroutine for the subsequent MST construction algorithms of [19, 29].

In this paper, we present an MST verification algorithm that achieves simultaneously $\tilde{O}(|E|)$ messages and $\tilde{O}(\sqrt{n}+D)$ time. This result appears to indicate that MST verification may be easier than MST construction. Conversely, we show that the verification problem is not much easier, by proving that the known lower bounds for MST construction also hold for the verification problem. Specifically, we show that $\Omega(|E|)$ messages must be sent in worst case by any MST verification algorithm, and that $\tilde{\Omega}(\sqrt{n}+D)$ communication rounds are also required.

Our $\Omega(|E|)$ lower bound on the number of messages is fairly straightforward. The $\tilde{\Omega}(\sqrt{n}+D)$ time lower bound is achieved by a (somewhat involved) modification of the corresponding lower bound for the computational task [28]. Our verification algorithm builds upon techniques taken from the time optimal MST construction paper [26], and from the papers on labeling schemes [20, 24][2]. More specifically, after collecting some topological structure at some central vertex (in a way inspired by [26]), the algorithm avoids spreading all this information to all vertices, in order to save on messages. Instead, it carefully divides this information into pieces using the flow labeling scheme of [20, 24] and sends one piece of information to each vertex. This distribution of information is done in such a way that allows the verification to carry on using these pieces of information only.

1.2 The Model

A point-to-point communication network is modeled as an undirected graph $G(V,E)$, where the vertices in V represent the network processors and the edges in E represent the communication links connecting them. The length of a path in G is the number of edges it contains. The *distance* between two vertices u and v is the length of the shortest path connecting them. The *diameter* of G, denoted D, is the maximum distance between any two vertices of G.

Vertices are assumed to have unique identifiers, and each vertex knows its own identifier. The vertices do not know the topology or the edge weights of the entire network, but they know the weights of the edges incident to them. More precisely, a weight function $\omega : E \to \mathbb{N}$ associated with the graph assigns a nonnegative integer weight $\omega(e)$ to each edge $e = (u,v) \in E$. The weight $\omega(e)$ is known to the adjacent vertices, u and v. We assume that the edge weights are bounded by a polynomial in n (the number of vertices). The

[2] The MST verification problem is considered in [24] from a different angle, closer to the notions of *local checking* [1, 3] or *computation with advice* [8, 11, 12]. Specifically, the focus therein is on the minimum size of a *label* (i.e., amount of information stored at a vertex) needed to allow verification of an MST in a single round, by exchanging the labels between neighboring vertices. The complexity of computing these labels are not in the scope of that paper.

vertices can communicate only by sending and receiving messages over the communication links. Each vertex can distinguish between its incident edges. Moreover, if vertex v sends a message to vertex u along the edge $e = (v, u)$, then upon receiving the message, vertex u knows that the message was delivered over the edge e.

Similarly to [16, 26], we assume that the communication is carried out in a synchronous manner, i.e., all the vertices are driven by a global clock. Messages are sent at the beginning of each round, and are received at the end of the round. (Clearly, our lower bounds hold for asynchronous networks as well.) At most one B-bit message can be sent on each link in each direction on every round. Similarly to previous work, we assume that $B = O(\log n)$. The model also allows vertices to detect the absence of a message on a link at a given round, which can be used to convey information. Hence at each communication round, a link can be at one of $2^B + 1$ possible states, i.e., it can either transmit any of 2^B possible messages, or remain silent.

1.3 The distributed MST Verification problem

Formally, the *minimum-weight spanning tree (MST) verification* problem can be stated as follows. Given a graph $G(V, E)$, a weight function ω on the edges, and a subset of edges $T \subseteq E$, referred as the *MST candidate*, it is required to decide whether T forms a minimum spanning tree on G, i.e., a spanning tree whose total weight $w(T) = \sum_{e \in T} \omega(e)$ is minimal. In the distributed model, the input and output of the *MST verification problem* are represented as follows. Each vertex knows the weights of the edges connected to its immediate neighbors. A degree-d vertex $v \in V$ with neighbors $u_1, \ldots, u_d$ has d *weight variables* $W_1^v, \ldots, W_d^v$, with W_i^v containing the weight of the edge connecting v to u_i, i.e., $W_i^v = \omega(v, u_i)$, and d *boolean indicator variables* $Y_1^v, \ldots, Y_d^v$ indicating which of the edges adjacent to v participate in the MST candidate that we wish to verify. The indicator variables must be consistent, namely, for every edge (u, v), the indicator variables stored at u and v for this edge must agree (this is easy to verify locally). Similarly to previous work on MST construction, we assume that the verification algorithm is initiated by a designated source node. Let T_Y be the set of edges marked by the indicator variables (i.e., all edges for which the indicator variable is set to 1). The output of the algorithm at each vertex v is an assignment to a (boolean) output variable A^v that must satisfy $A^v = 1$ if T_Y is an MST of $G(V, E, \omega)$, and $A^v = 0$ otherwise.

1.4 Our Results

We establish asymptotically tight bounds for the time and message complexities of the MST verification problem. Specifically, in the positive direction we show the following:

▶ **Theorem 1.1.** *There exists a distributed MST verification algorithm that uses $\tilde{O}(\sqrt{n}+D)$ time and $\tilde{O}(|E|)$ messages.*

This upper bound is complemented by two lower bounds.

▶ **Theorem 1.2.** *Any distributed algorithm for MST verification requires $\Omega(|E|)$ messages.*

▶ **Theorem 1.3.** *Any distributed algorithm for MST verification requires $\tilde{\Omega}(\sqrt{n} + D)$ time.*

Theorem 1.2 is proved (in a rather straightforward manner) assuming that a vertex knows only its own identifier. We note, however, that it can be generalized to the model where each vertex knows also the identifiers of its neighbors, yielding $\Omega(|E|)$ lower bounds similar to those of [4]. Due to lack of space, the proof of Theorem 1.3 is deferred to the full paper.

2 An MST Verification Algorithm

2.1 Definitions and Notations

Following are some definitions and notations used in the description of the algorithm. For a graph $G = (V, E, w)$, an edge e is said to be *cycle-heavy* if there exists a cycle C in G that contains e, and e has the heaviest weight in C. For a graph $G = (V, E, w)$, a set of edges $F \subseteq E$ is said to be an *MST fragment* of G if there exists a minimum spanning tree T of G such that F is a subtree of T (i.e., $F \subseteq T$ and F is a tree). Similarly, a collection $\mathcal{F}$ of edge sets is referred to as an *MST fragment collection* of G if there exists an MST T of G such that (1) F_i is a subtree of T for every $F_i \in \mathcal{F}$, (2) $\bigcup\limits_{F_i \in \mathcal{F}} V(F_i) = V$, and (3) $V(F_i) \cap V(F_j) = \emptyset$ for every $F_i, F_j \in \mathcal{F}$.

Consider a graph $G = (V, E, w)$, an MST fragment collection $\mathcal{F}$, a subgraph T of G and a vertex v in G. The *fragment graph* of G, denoted $G_{\mathcal{F}}$, is defined as a graph whose vertices are the MST fragments $F_i \in \mathcal{F}$, and whose edge set contains an edge (F_i, F_j) iff there exist vertices $u \in V(F_i)$ and $v \in V(F_j)$ such that $(u, v) \in E$. The *fragment graph induced by* T, denoted $T_{\mathcal{F}}$, is defined as a graph whose vertices are the MST fragments $F_i \in \mathcal{F}$, and whose edge set contains an edge (F_i, F_j) iff there exist vertices $u \in V(F_i)$ and $v \in V(F_j)$ such that $(u, v) \in T$. The edges of $T_{\mathcal{F}}$ are also referred to as the *inter-cluster* edges induced by T. For each fragment $F_i \in \mathcal{F}$, the set of *fragment internal* edges induced by T, denoted T_I^i, consists of all edges of T with both endpoints in $V(F_i)$, i.e., $T_I^i = \{e \mid e = (u, v) \in T \textit{ and } \; u, v \in V(F_i)\}$. The fragment of v, denoted by $\mathcal{F}(v)$, is the fragment $F_i \in \mathcal{F}$ such that $v \in V(F_i)$. Denote by $E_{\mathcal{F}}(v)$ the set of fragment internal edges that are incident to v (i.e., $E_{\mathcal{F}}(v) = \{e \mid \; e = (u, v) \in E \textit{ and } u \in V(\mathcal{F}(v))\}$). Similarly, denote by $E_{T_I}(v)$ the set of fragment internal edges induced by T and incident to v.

Throughout the description of the verification algorithm we assume that the edge weights are distinct. Having distinct edge weights simplifies our arguments since it guarantees the uniqueness of the MST. Yet, this assumption is not essential. Alternatively, in case the graph is not guaranteed to have distinct edge weights, we may use a modified weight function $\omega'(e)$, which orders edge weights lexicographically: first, by their original weight $\omega(e)$, then, by the indicator variable of the edge, and finally, by the identifiers of the edge endpoints. Under the weight function $\omega'(e)$, edges with indicator variable set to "true" will have lighter weight than edges with the same weight under $\omega(e)$ but with indicator variable set to "false" (i.e., for edges $e_1 \in T$ and $e_2 \notin T$ such that $w(e_1) = w(e_2)$, we have $w'(e_1) < w'(e_2)$). It follows that the given subgraph T is an MST of $G(V, E, \omega)$ iff T is an MST of $G(V, E, \omega')$. Moreover, since $\omega'(\cdot)$ takes into account the unique vertex identifiers, it assigns distinct edge weights.

The MST verification algorithm makes use of Procedures `DOM_Part` and `MAX_Label`, presented in [26] and [20] respectively. The distributed Procedure `DOM_Part`, used in [26], partitions a given graph into an *MST fragment collection (MFC)* $\mathcal{F}$, where each fragment is of size at least $k + 1$ and depth $O(k)$, for a specified parameter k. A *fragment leader* vertex is associated with each constructed fragment (the identifier of the fragment is the identifier of the fragment's leader). After Procedure `DOM_Part` is completed, each vertex v knows the identifier of the fragment to which it belongs and v's incident edges that belong to the fragment. (To abide by the assumption of [26] that each vertex knows the identifiers of its neighbors, before applying Procedure `DOM_Part`, the algorithm performs a single communication round that exchanges vertex identifiers between neighboring vertices.)

The labeling scheme `MAX_Label` of [20] which is designed for the family of weighted trees constructs an encoder algorithm $\mathcal{E}$ and a decoder algorithm $\mathcal{D}$ that satisfy the following: **1.** Given a weighted tree T, the encoder algorithm $\mathcal{E}$ assigns a label $L(v)$ to each vertex v of T.

2. Given two labels $L(u), L(v)$ assigned by $\mathcal{E}$ to two vertices u and v in some weighted tree T, the decoder algorithm $\mathcal{D}$ outputs $MAX(u, v)$, which is the maximum weight of an edge on the path connecting u and v in T. (The decoder algorithm $\mathcal{D}$ bases its answer on the pair of labels $L(u), L(v)$ only, without knowing any additional information regarding the tree T.) The labeling scheme `MAX_Label` produces $O(\log n \log W)$-bit labels, where W is the largest weight of an edge. Since W is assumed to be polynomial in n, the label size is $O(\log^2 n)$ bit.

2.2 The algorithm

The algorithm consists of three phases. The first phase starts by running the distributed Procedure `DOM_Part` of [26], which constructs an MST fragment collection (MFC) $\mathcal{F}$ for a fixed parameter k that will be specified later. The algorithm verifies that the set of edges contained in the constructed MFC is equal to the set of fragment internal edges induced by the MST candidate T, namely, $\underset{F_i \in \mathcal{F}}{\cup} F_i = \underset{F_i \in \mathcal{F}}{\cup} T_I^i$ (note that this is a necessary condition for correctness since the graph is assumed to have a unique MST).

In the second and third phases, the algorithm verifies that all remaining edges of T form an MST on the fragment graph $G_{\mathcal{F}}$. Let $T_{\mathcal{F}}$ be the fragment graph induced by T with respect to the MFC $\mathcal{F}$ found in the previous phase. In order to verify that $T_{\mathcal{F}}$ forms an MST on the fragment graph $G_{\mathcal{F}}$, it suffices to verify that $T_{\mathcal{F}}$ is a tree and that none of the edges of $T_{\mathcal{F}}$ is a *cycle heavy* edge in $G_{\mathcal{F}}$. The above is done by using the labeling scheme ($\mathcal{E}$,$\mathcal{D}$) of [20] (or [24]) on $T_{\mathcal{F}}$. Informally, the algorithm assigns a label $L(F_i)$ to each vertex F_i of $T_{\mathcal{F}}$ using the encoder algorithm $\mathcal{E}$ applied on $T_{\mathcal{F}}$. The label $L(F_i)$ is then efficiently delivered to each vertex in F_i. Recall, that given the labels of two fragments $L(F_i), L(F_j)$ it is now possible to compute the weight of the heaviest edge on the path connecting the fragments in $T_{\mathcal{F}}$ by applying the decoder algorithm $\mathcal{D}$. Once all vertices obtain the labels of their corresponding fragments, each vertex of G can verify (using the decoder $\mathcal{D}$) that each inter fragment edge incident to it and not participating in $T_{\mathcal{F}}$ forms a cycle when added to $T_{\mathcal{F}}$ for which it is a cycle heavy edge. Following is a more detailed description of the algorithm.

1. a. The source vertex s (that initiates the algorithm) constructs a BFS tree rooted at s, computes the values n and D and broadcasts a signal on the BFS tree instructing each vertex to send its identifier to all its neighbors.
   b. Perform Procedure `DOM_Part(`k`)`, where k is specified later. (The result is an MFC $\mathcal{F}$, where each fragment $F \in \mathcal{F}$ is of size $|V(F)| > k$ and depth $O(k)$, having a fragment leader and a distinct fragment identifier known to all vertices in the fragment).
   c. Each vertex sends its fragment identifier to all its neighbors.
   d. Each vertex v identifies the sets of edges $E_{\mathcal{F}}(v)$ and $E_{T_I}(v)$.
   e. Verify that $\underset{F_i \in \mathcal{F}}{\cup} F_i = \underset{F_i \in \mathcal{F}}{\cup} T_I^i$ by verifying at each vertex v that $E_{\mathcal{F}}(v) \subseteq T$ and $E_{T_I}(v) \subseteq \mathcal{F}(v)$. (Else return "$T$ is not an MST".)
2. a. Vertex s counts the number of fragments by performing a convergecast on the BFS tree (only fragment leader vertices increase the counter of the convergecast). Let f be the number of fragments.
   b. Vertex s counts the number of inter fragment edges induced by T (i.e., the number of edges in $T_{\mathcal{F}}$) by performing a convergecast on the BFS tree. Then, Vertex s verifies that the number of edges is equal to $f - 1$. (Else return "T is not an MST".)
   c. All vertices send the description of all edges in $T_{\mathcal{F}}$ to s, by performing an upcast on the BFS tree. (The edges of $T_{\mathcal{F}}$ are all edges of T that connect vertices from different fragments.)

   d. Vertex s verifies that $T_{\mathcal{F}}$ is a tree. (Else return "T is not an MST".)

3. **a.** Each fragment leader sends a message to vertex s over the BFS tree. Following these messages, all vertices save routing information regarding the fragment leaders. I.e., if v is a fragment leader and v is a descendant of some other vertex u in the BFS tree, then, after this step is applied, vertex u knows which of its children is on the path connecting it to the fragment leader v.

   b. Vertex s computes the labels $L(F_i)$ for all vertices F_i in $T_{\mathcal{F}}$ (i.e., assigns a label to each of the fragments) using the encoder algorithm $\mathcal{E}$. Subsequently, vertex s sends to each fragment leader its fragment label (the label of each fragment is sent to the fragment leader over the BFS tree using the routing information established in Step 3a above). Recall, each label is encoded using $O(\log^2 n)$ bits.

   c. Each fragment leader broadcasts its fragment label to all vertices in the fragment. The broadcast is done over the fragment edges.

   d. Each vertex sends its fragment label to all its neighbors.

   e. Each vertex v verifies for every neighbor u not belonging to v's fragment, and s.t. $(u, v) \notin T$, that $w(v, u) \geq MAX(\mathcal{F}(v), \mathcal{F}(u))$ (the value $MAX(F(v), F(u))$ is computed by applying the decoder algorithm $\mathcal{D}$ on labels $L(\mathcal{F}(v)), L(\mathcal{F}(u)))$. (Else return "$T$ is not an MST".)

2.3 Complexity

Steps 1a and 1c clearly take $O(D)$ time and $O(|E|)$ messages. Step 1b, i.e., the execution of Procedure `DOM_Part`, requires $O(k \cdot \log^* n)$ time and $O(E \cdot \log k + n \cdot \log^* n \cdot \log k)$ messages. (A full analysis appears in [30].) The remaining steps of the first phase are local computations performed by all vertices. Thus, the first phase of the MST verification algorithm requires $O(D + k \cdot \log^* n)$ time and $O(E \cdot \log k + n \cdot \log^* n \cdot \log k)$ messages.

Since the fragments are disjoint and each fragment contains at least k vertices, the number of MST fragments constructed during the first phase of the algorithm is $f \leq n/k$. The table below summarizes the time and message complexities of the second and third phases of the algorithm.

Step	Description	Time	Messages
2a,2b	BFS convergecast	$O(D)$	$O(\|E\|)$
3d	Communication between neighbors	$O(\log n)$	$O(\log n \cdot \|E\|)$
2d,3e	Local computation	0	none
2c,3a	BFS upcast of f messages	$O(D + f)$	$O(f \cdot D)$
3b	BFS downcast of f messages (each of size $\log^2 n$)	$O(D + f \cdot \log n)$	$O(f \cdot \log n \cdot D)$
3c	Broadcast in each of the MST fragments	$O(k + \log n)$	$O(\log n \cdot n)$

Combining the above arguments, we get that the algorithm requires time $O(\frac{n}{k} \cdot \log n + k \cdot \log^* n + D)$ and $O(E \cdot \log n + n \cdot \log^* n \cdot \log k + \frac{n}{k} \cdot \log n \cdot D + n \cdot \log n)$ messages. Recall that after Step 1a is applied, the source vertex s knows the number of nodes n and the diameter D. By choosing $k = \sqrt{n}$ in case $D < \sqrt{n}$ and $k = D$ otherwise, we get the following, implying Theorem 1.1.

▶ **Lemma 2.1.** *The MST verification algorithm requires* $O(\sqrt{n} \cdot \log n + D)$ *time and* $(|E| \cdot \log n)$ *messages.*

2.4 Correctness

We now show that our MST verification algorithm correctly identifies whether the given tree T is an MST. We begin with the following claim.

▶ Claim 2.2. Let T be a spanning tree of G such that T contains all edges of the MFC $\mathcal{F}$ and $T_{\mathcal{F}}$ forms an MST on the fragment graph of G (with respect to the MFC $\mathcal{F}$). Then T is an MST on G.

Proof. Since $\mathcal{F}$ is an MST Fragment collection, there exists an MST T' of G such that T' contains all edges of $\mathcal{F}$. Due to the minimality of T', the fragment graph $T'_{\mathcal{F}}$ induced by T' necessarily forms an MST on the fragment graph of G (with respect to the MFC $\mathcal{F}$). Hence we get that $w(T) = w(T')$, thus T is an MST of G. ◀

Due to the assumption that edge weights are distinct, we get:

▶ Observation 2.3. The MST of G is unique.

By combining Claim 2.2 and Observation 2.3 we get the following.

▶ Claim 2.4. A spanning tree T of G is an MST if and only if T contains all edges of the MFC $\mathcal{F}$ and $T_{\mathcal{F}}$ forms an MST on the fragment graph of G (with respect to the MFC $\mathcal{F}$)

▶ **Lemma 2.5.** *The MST verification algorithm correctly identifies whether the given tree T is an MST of the graph G.*

Proof. By Claim 2.4, to prove the correctness of the algorithm it suffices to show that given an MST candidate T, the algorithm verifies that:
(1) T is a spanning tree of G,
(2) T contains all edges of $\mathcal{F}$, and
(3) $T_{\mathcal{F}}$ forms an MST on the fragment graph of G with respect to the MFC $\mathcal{F}$.

Since $\mathcal{F}$ as constructed by Procedure `DOM_Part` in the first phase is an MFC, it spans all vertices of G. Step 1e verifies that $\bigcup_{F_i \in \mathcal{F}} F_i = \bigcup_{F_i \in \mathcal{F}} T^i_I$, thus after this step, it is verified that T does not contain a cycle that is fully contained in some fragment $F_i \in \mathcal{F}$ (since every $F_i \in \mathcal{F}$ is a tree). On the other hand, step 2d verifies that T does not contain a cycle that contains vertices from different fragments. Hence, the algorithm indeed verifies that T is a spanning tree of G, and Property (1) follows. Property (2) is clearly verified by step 1e of the algorithm. Finally, to verify that $T_{\mathcal{F}}$ forms an MST on the fragment graph of G it suffices to verify that inter-fragment edges not in $T_{\mathcal{F}}$ are cycle heavy, which is done in step 3e. Property (3) follows. ◀

3 Message Complexity Lower Bound

We prove a message complexity lower bound of $\Omega(|E|)$ on the *Spanning Tree (ST) verification* problem, which is a relaxed version of the MST verification problem defined as follows. Given a graph $G = (V, E, \omega)$ and a subgraph T (referred to as the *ST candidate*), we wish to decide whether T is a spanning tree of G (not necessarily of minimal weight). Clearly, a lower bound on *ST verification* problem also applies to the *MST verification* problem. Since spanning tree verification is independent of the edge weights, for convenience we consider unweighted networks throughout the lower bound proof.

We begin with a few definitions. A *protocol* is a local program executed by all vertices in the network. In every step, each processor performs local computations, sends and receives messages, and changes its state according to the instructions of the protocol. A protocol achieving a given task should work on every network G and every ST candidate T. Following [4], we denote the *execution* of protocol Π on network $G(V, E)$ with ST candidate T by $EX(\Pi, G, T)$. The *message history* of an execution $EX = EX(\Pi, G, T)$ is a sequence describing the messages sent during the execution EX. Consider a protocol Π, two graphs $G_0(V, E_0)$ and $G_1(V, E_1)$ over the same set of vertices V, and two ST candidates T_0 and T_1 for G_0 and G_1 respectively, and the corresponding executions $EX_0 = EX(\Pi, G_0, T_0)$ and $EX_1 = EX(\Pi, G_0, T_1)$. We say that the executions are *similar* if their message history is identical.

Let $G = (V, E)$ be a graph (together with an assignment of vertex identifiers), T be a subgraph and $e = (u, v)$ be one of its edges. Let $G' = (V', E')$ be some copy of $G = (V, E)$, where the identifiers of the vertices in V' are not only pairwise distinct but also distinct from the given identifiers on V. Consider the following graphs G^2 and G^e and the subgraph T^2.
• Graph G^2 is simply $G^2 = (V^2, E^2) = G \cup G' = (V \cup V', E \cup E')$. The subgraph T^2 of G^2 is defined as the union of the two copies of T, one in G and the other in G'.
• The graph G^e is a "cross-wired" version of G^2. Formally, $G^e = (V^2, E^e)$, where $E^e = E^2 \smallsetminus \{(u, v), (u', v')\} \cup \{(u, v'), (u', v)\}$. (Observe, for $e \notin T$, T^2 is also a subgraph of G^e.)

Let Π be a protocol that correctly solves ST verification problem. Fix G to be some arbitrary graph, fix a copy G' of G, fix a spanning tree T of G, and fix a source vertex $s \in V$ initiating the execution of Π on either of the graphs G, G^2 and G^e with the ST candidates T, T^2 and T^2, respectively. We stress that G^2 (with candidate T^2) is not a valid input for the ST (or the MST) verification problem since it is not connected. Still, we can consider the execution $EX(\Pi, G^2, T^2)$, without requiring anything from its output.

▶ **Lemma 3.1.** *Let $e \in E \setminus E(T)$, such that no message is sent over the edges e and e' in execution $EX(\Pi, G^2, T^2)$. Then executions $EX(\Pi, G^2, T^2)$ and $EX(\Pi, G^e, T^2)$ are similar.*

Proof. Proof Sketch of lemma 3.1. We show that in both executions each vertex sends and receives identical sequences of messages in each communication round of the protocol. Note that at each round the messages sent by some vertex w is dependent on w's topological view (neighbors of w), w's initial input (the indicator variables of the edges incident to w), and the set of messages sent and received by w in previous communication rounds. Denote by EX^2 and EX^e executions $EX(\Pi, G^2, T^2)$ and $EX(\Pi, G^e, T^2)$ respectively. Note that any vertex $w \in V^2 \smallsetminus \{u, v, u', v'\}$ has identical topological view and identical initial input in both executions. Vertex u has identical initial input and identical number of neighbors in both executions. Though the communication link connecting u to v in G^2 connects u to v' in G^e, vertex u is initially unaware of this difference between the executions since it does not know the identifiers of its neighbors. (Same holds for vertices v, u' and v'.) The proof is by induction on r, the number of communication rounds of protocol Π.
Induction base: For $r = 0$. In the first communication round, the messages sent by each vertex depend solely on its topological view and initial input. Let us analyze the sequence of messages sent by vertices in V (the vertices of graphs G^2 and G^e that belong to the first copy of G). Following are the possible cases.
Vertex $w \notin \{u, v\}$: Vertex w has identical topological view and identical initial input in both execution, thus it sends identical sequence of messages in the first round of both executions.
Vertex u: As mentioned above, although in execution EX^e vertex u is connected to v' instead of v, it has no knowledge of this difference. Thus u sends identical sequence of messages over each of its communication links. The fact that no messages are sent over edge e in execution EX^2, implies that in execution EX^e no message is sent by u to its neighbor v'. Thus, u sends identical sequence of messages in the first communication round of both

executions.
Vertex v: can be analyzed in the same manner as vertex u.
The above shows that vertices in V send the same sequence of messages in the first communication round of both executions. The induction base claim follows by applying the same argument on vertices of V'.

Inductive step: Can be shown using a similar case analysis as in the induction base. ◀

Theorem 1.2 follows as a consequence of the following Lemma.

▶ **Lemma 3.2.** *Execution $EX(\Pi, G^2, T^2)$ requires $\Omega(|E^2 \setminus T^2|)$ messages.*

Proof. Assume, towards contradiction, that there exists a protocol Π that correctly solves the ST verification problem for every graph G and ST candidate T, such that execution $EX(\Pi, G, T)$ sends fewer than $|E \setminus T|/2$ messages over edges from $E \setminus T$.

For the rest of the proof we fix $G = (V, E)$ to be an arbitrary connected graph and denote the ST candidate by T. We take T to be a spanning tree and not just any subgraph. (See Figure 1).

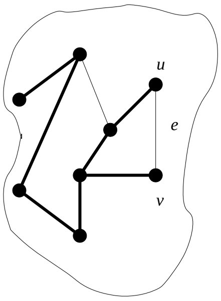


Figure 1 Graph G with ST candidate T (the bold edges belong to T)

Consider the graph G^2 as previously defined with ST candidate $T^2 = \{e \in T\} \cup \{(u', v') : e = (u, v) \in T\}$ (See Figure 2).

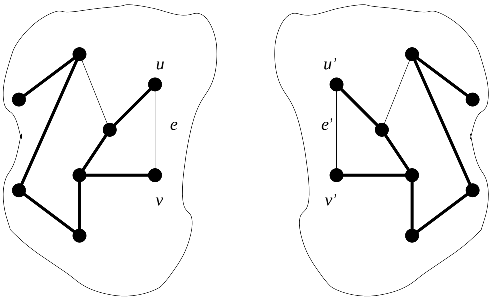


Figure 2 Graph G^2 with ST candidate T^2 (the bold edges belong to T^2)

Then by the assumption on Π, execution $EX^2 = EX(\Pi, G^2, T^2)$ sends fewer than $|E^2 \setminus T^2|/2$ messages over edges from $E^2 \setminus T^2$. Hence there exist $e = (u, v)$ and $e' = (u', v')$ such that $e, e' \in E^2 \setminus T^2$ and no message is sent over e and e' in execution EX^2. Consider the graph G^e with ST candidate T^2 as previously defined (See Figure 3).

By Lemma 3.1, executions EX^2 and $EX^e = EX(\Pi, G^e, T^2)$ are similar. Note that $e, e' \notin T^2$, thus T^2 is not a spanning tree of G^e (since the two copies of G contained in G^e

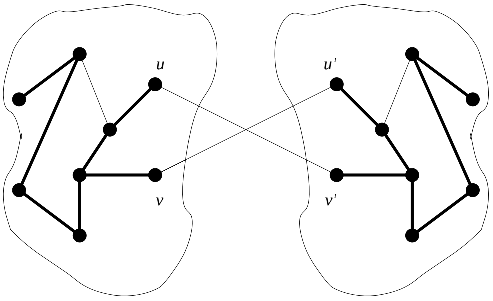


Figure 3 Graph G^e with ST candidate T^2 (the bold edges belong to T^2)

are connected solely by edges e and e'). Since Π correctly solves the ST verification problem, the output of all vertices in EX^e is "0" (i.e., the given ST candidate T^2 is not a spanning tree of the graph G^e). On the other hand, consider the execution $EX = (\Pi, G, T)$ with ST candidate T. Note that EX is exactly the restriction of EX^2 on the first copy of G contained in G^2. Since G^2 contains two disconnected copies of G the output of all vertices in execution EX^2 will be identical to the output of the same vertices in EX (since in both executions the vertices have identical topological view and the input variables contain identical values). Since executions EX^2 and EX^e are similar, the output of EX is "0", in contradiction to the correctness of Π. ◀

References

1 Y. Afek, S. Kutten, and M. Yung. The local detection paradigm and its applications to self stabilization. *Theoretical Computer Science*, 186(1-2):199–230, 1997.

2 B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Proc. 19th ACM Symp. on Theory of computing (STOC)*, pages 230–240, New York, NY, USA, 1987. ACM.

3 B. Awerbuch and G. Varghese. Distributed program checking: a paradigm for building self-stabilizing distributed protocols. In *Proc. IEEE Symp. on Foundations of Computer Science (FOCS)*, pages 258–267, 1991.

4 Baruch Awerbuch, Oded Goldreich, Ronen Vainish, and David Peleg. A trade-off between information and communication in broadcast protocols. *J. ACM*, 37(2):238–256, 1990.

5 Adam L. Buchsbaum, Loukas Georgiadis, Haim Kaplan, Anne Rogers, Robert E. Tarjan, and Jeffery R. Westbrook. Linear-time algorithms for dominators and other path-evaluation problems. *SIAM Journal on Computing*, 38(4):1533–1573, 2008.

6 James E. Burns. A formal model for message passing systems. Technical Report TR-91, Computer Science Dept., Indiana University, Bloomington, 1980.

7 I. Cidon, I. Gopal, M. Kaplan, and S. Kutten. A distributed control architecture of high-speed networks. *IEEE Trans. on Communications*, 43(5):1950–1960, 1995.

8 R. Cohen, P. Fraigniaud, D Ilcinkas, A Korman, and D. Peleg. Label-guided graph exploration by a finite automaton. *ACM Trans. on Algorithms*, 4(4), 2008.

9 B. Dixon, M. Rauch, and R.E. Tarjan. Verification and sensitivity analysis of minimum spanning trees in linear time. *SIAM Journal on Computing*, 21(6):1184–1192, 1992.

10 B. Dixon and R.E. Tarjan. Optimal parallel verification of minimum spanning trees in logarithmic time. *Algorithmica*, 17(1):11–18, 1997.

11 P. Fraigniaud, D Ilcinkas, and A. Pelc. Communication algorithms with advice. *J. Comput. Syst. Sci.*, 76(3-4):222–232, 2008.

12 P. Fraigniaud, A Korman, and E. Lebhar. Local mst computation with short advice. In *Proc. SPAA*, pages 154–160, 2007.

13 Greg N. Frederickson and Nancy A. Lynch. The impact of synchronous communication on the problem of electing a leader in a ring. In *Proc. 16th ACM Symp. on Theory of computing (STOC)*, pages 493–503, New York, NY, USA, 1984. ACM.

14 M. L. Fredman and D. E. Willard. Trans-dichotomous algorithms for minimum spanning trees and shortest paths. *Proc. 31st IEEE FOCS*, pages 719–725, 1990.

15 R. G. Gallager, P. A. Humblet, and P. M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Trans. Program. Lang. Syst.*, 5(1):66–77, 1983.

16 J. Garay, S. A. Kutten, and D. Peleg. A sub-linear time distributed algorithm for minimum-weight spanning trees. *SIAM Journal on Computing*, 27(1):302–316, 1998.

17 R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *Ann. Hist. Comput.*, 7(1):43–47, 1985.

18 D. Harel. A linear time algorithm for finding dominators in flow graphs and related problems. In *Proc. 17th ACM Symp. on Theory of computing (STOC)*, pages 185–194, 1985.

19 D.R Karger, P.N. Klein, and R. E. Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42(2):321–328, 1995.

20 M. Katz, N.A. Katz, A. Korman, and D. Peleg. Labeling schemes for flow and connectivity. *SIAM J. Comput.*, 34(1):23–40, 2005.

21 V. King. A simpler minimum spanning tree verification algorithm. *Algorithmica*, 18:263–270, 1997.

22 V. King, C.K Poon, V Ramachandran, and S. Sinha. An optimal erew pram algorithm for minimum spanning tree verification. *Information Processing Letters*, 62(3):153–159, 1997.

23 J. Komlòs. Linear verification for spanning trees. *Combinatorica*, 5:57–65, 1985.

24 A. Korman and S. Kutten. Distributed verification of minimum spanning trees. *Distributed Computing*, 20(4):253–266, 2007.

25 A. Korman, S. Kutten, and D Peleg. Proof labeling schemes. *Distributed Computing*, 22(4):215–233, 2010.

26 Shay Kutten and David Peleg. Fast distributed construction of small k-dominating sets and applications. *J. Algorithms*, 28(1):40–66, 1998.

27 Zvi Lotker, Boaz Patt-Shamir, and David Peleg. Distributed mst for constant diameter graphs. In *Proc. 20th ACM Symp. on Principles of distributed computing (PODC)*, pages 63–71, New York, NY, USA, 2001. ACM.

28 D. Peleg and V. Rubinovich. A near-tight lower bound on the time complexity of distributed minimum-weight spanning tree construction. *SIAM J. Comput.*, 30(5):1427–1442, 2000.

29 S. Pettie and V. Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49(1):16–34, 2002.

30 Vitaly Rubinovich. Distributed minimum spanning tree construction. Master's thesis, Department of Computer Science and Applied Mathematics, The Weizmann Institute of Science, 1999.

31 R.E. Tarjan. Applications of path compression on balanced trees. *J. ACM*, 26:690–715, 1979.

32 R.E. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, 1983.

Automata based verification over linearly ordered data domains*

Luc Segoufin[1] **and Szymon Toruńczyk**[†] [2]

1 **INRIA and ENS Cachan, LSV**
2 **University of Warsaw**

Abstract

In this paper we work over linearly ordered data domains equipped with finitely many unary predicates and constants. We consider nondeterministic automata processing words and storing finitely many variables ranging over the domain. During a transition, these automata can compare the data values of the current configuration with those of the previous configuration using the linear order, the unary predicates and the constants.

We show that emptiness for such automata is decidable, both over finite and infinite words, under reasonable computability assumptions on the linear order.

Finally, we show how our automata model can be used for verifying properties of workflow specifications in the presence of an underlying database.

1998 ACM Subject Classification F.4 Mathematical Logic and Formal Languages

Keywords and phrases Register automata, data values, linear order

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.81

1 Introduction

System verification often requires dealing with infinite state systems. There are many sources of infinity, one of them being the presence of variables ranging over an infinite set of data values and this is the focus of this paper.

There exist several decidable models of automata and logics that explicitly manipulate data values and that can be used for verification. In order to achieve decidability there is a necessary trade-off between the permitted operations on data and the allowed recursion. For instance, many models consider only equality tests between data values [3, 10, 14], or limit the recursion or the expressive power [4, 5, 8, 9], or only apply over specific data domains [11, 7, 8, 9, 6, 1].

In terms of possible operation on data values, equality tests permit already a wide range of recursion schemes and the corresponding decidability results can be used for modeling a variety of applications. However it has been advocated in [11] that comparisons based on a linear order over the data values could be useful in many scenarios, including data centric applications built on top of a database. They propose a model for specifying "artifact centric workflows" in the presence of a database and prove that temporal properties can be verified in PSPACE, if the data domain is the set of rational numbers.

In this paper, we consider automata over words which are equipped with a finite set of variables, ranging over a linearly ordered structure. Transitions of the automaton are based

* This research was funded by the ERC research project FoX under grant agreement FP7-ICT-233599.
† Thanks for funding from ESF AutoMathA.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 81–92
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

on constraints on the variables using the vocabulary of the structure. We present a method for analyzing emptiness of such automata over finite and infinite words, independently of the linearly ordered structure. We derive from it several useful decidability results. Below we describe these contributions in more details.

The setting. We consider arbitrary linearly ordered structures. Apart for a linearly ordered set, the structure may be equipped with finitely many unary predicates and constants. Over the integers, typical unary predicates might denote the set of primes or the set of numbers divisible by a fixed constant.

Automata. We present a model of automata (either for finite or infinite words) over such linearly ordered structures. A configuration of the automaton is a tuple of data values of a fixed arity. A transition constrains the values of the next configuration relative to the values of the current one by a Boolean combination of predicates in the vocabulary of the structure. The initial and accepting configurations are also specified using similar constraints.

The potential. Our main contribution is a generic toolbox for the described model of automata, which is applicable to all linearly ordered structures. It is based on the notions of *potential* and *saturation*. Intuitively, saturation[1] transforms a linearly ordered structure by inserting finitely many new constants until all intervals between two consecutive constants contain infinitely many or no points of a certain property. Once the structure is saturated, it admits a potential. One can think of the potential as of a measure of how generic a configuration is; the main property is that an automaton may choose the next configuration to be sufficiently generic, provided the previous configuration was also generic. The rest of the paper can be seen as applications of these notions.

Main results. As a first application of our toolbox, we show that over any linearly ordered structure our automata can be simulated by finite state automata. This implies that emptiness for our automata model is decidable for finite and infinite runs, if the structure satisfies reasonable complexity assumptions, typically being able to test for satisfiability of a set of constraints expressed using the predicates of the structure. This yields PSpace decision procedures for many linearly ordered structures, in particular integers and rationals.

We also present a variant of LTL, where Boolean predicates are replaced with constraints using the predicates of the structure, and which works over words extended by tuples of data values. This logic can be translated into our automata model. Combining this with the emptiness test mentioned above, we obtain decidability results concerning this logic and PSpace complexity in most important cases. As our automata are closed under intersection, this allows to test LTL properties on the runs of a given automata.

Finally, our last result shows how our toolbox can be used for dealing with automata that moreover have the possibility of consulting a finite database in order to constrain their transitions. This method works for all linearly ordered structures, giving an alternative proof of the result of [11] over rationals but also solving the case of integers, which was posed as an open problem.

Related work. Our automata model is very close to the one used in [7] over integer data values. Čerāns solved the decidability of the emptiness problem using Dickson's lemma. The obtained decision algorithm is therefore non-primitive recursive. That proof does not apply to dense linear orders and it's not clear how it can be extended to incorporate the presence of an underlying database. These issues are solved in this paper.

Our model of automata generalizes the register automata studied in [6, 14] – indeed,

[1] Our notion of saturation differs from the usual logical one in two important ways: it considers only existential types and it is constructive.

a configuration can be seen as a valuation of the registers and additionally of the current datum of the input word. Our model is more expressive as we can compare data values using a linear order and unary predicates. Our setting also generalizes the model of [1] for verifying properties of programs working on arrays. Their model allows for linear tests but is specific to integers and finite runs.

Our notion of potential uses a partitioning of the space into cells. Similar techniques were developed for timed automata or over dense linearly ordered structures. See for instance [12]. Over dense linearly ordered structures, register automata generalize the notion of timed automata in a sense explained in [13]. Our notion of potential hence generalizes these ideas even for discrete linearly ordered structures.

The work of [11] considers specification and verification of workflows over finite databases whose underlying domain is the set of rationals. In fact, over the rationals, our model of automata can be viewed as a simplification of the elaborate formalism used in [11], necessary for the specific application targeted therein. As is shown in that paper, LTL formulas can be checked in PSpace assuming a fixed database schema, ExpSpace otherwise. We obtain similar results but our proof also applies for other linearly ordered structures, in particular over the integers, settling an open problem raised in [11].

There exist many extensions of LTL with several kinds of constraints over various data domains. Decidable fragments can compare data values using their relative order or their value modulo some constant. The complexity of satisfiability is shown to be PSpace-complete, see [8] for a survey. This also follows from our results.

Due to space limitations many proofs are omitted or only sketched. They will appear in the journal version of this paper. We thank the anonymous referees for the useful comments.

2 Preliminaries

Linearly ordered structures. By a *linearly ordered structure* we refer to a structure of the form $\mathcal{D} = \langle D, <, P_1, P_2, \ldots, P_l, c_1, c_2, \ldots, c_m \rangle$, where D is the domain of the structure, $<$ is a linear order on D, $P_1, \ldots, P_l$ are unary predicates denoting subsets of D and $c_1, \ldots, c_m$ are constants. Typical examples are $\langle \mathbb{Z}, <, 0, 1 \rangle$, $\langle \mathbb{Q}, <, 0 \rangle$, $\langle \{a, b\}^*, <_{lex}, P_a, P_b, \epsilon \rangle$ where $<_{lex}$ is the lexicographical order and P_a, P_b are unary predicates indicating the last letter of a word. But we also consider more elaborate linear orders such as $\langle \mathbb{Q}, <, P_{even}, P_{odd}, P_{prime} \rangle$, where the three predicates correspond to even integers, odd integers and prime integers.

Formulas and cells. We now assume a fixed linearly ordered structure $\mathcal{D}$ and dimension k. We consider k-ary relations over the domain of $\mathcal{D}$ definable by a Boolean combination of atoms built from the symbols of $\mathcal{D}$ using k variables. Each set of this form will be called a *region in* $\mathcal{D}^k$. A region that corresponds to a maximal consistent conjunction of atoms or negated atoms is called a *cell in* $\mathcal{D}^k$. Note that there are finitely many cells and any region is a disjoint union of cells. For instance, over $\mathcal{D} = \langle \mathbb{Z}, <, 0 \rangle$, $x < y$ defines a region of $\mathcal{D}^2$ which is the disjoint union of 5 cells: $x < y < 0$, $x < y = 0$, $x < 0 < y$, $x = 0 < y$ and $0 < x < y$. A tuple $\bar{x} \in D^k$ will be also called a *point in* $\mathcal{D}^k$. For each such $\bar{x}$ we denote by $\langle \bar{x} \rangle$ the unique cell in $\mathcal{D}^k$ containing $\bar{x}$.

We fix a finite alphabet A. A *$\mathcal{D}$-automaton $\mathcal{A}$ of dimension k* is a tuple $((\delta_a)_{a \in A}, \tau_I, \tau_F)$ described as follows. For each letter $a \in A$, δ_a is a region in $\mathcal{D}^k \times \mathcal{D}^k = \mathcal{D}^{2k}$, representing the allowed transitions of $\mathcal{A}$ when reading the letter a; τ_I and τ_F are regions in $\mathcal{D}^k$ denoting the initial and accepting configurations of $\mathcal{A}$.

The configurations of $\mathcal{A}$ are points in $\mathcal{D}^k$. Let $w = a_1 a_2 \ldots a_n$ be a finite word. A *run* ρ of $\mathcal{A}$ on w is a sequence of configurations $\bar{x}_0, \bar{x}_1, \ldots, \bar{x}_n \in D^k$ such that $\bar{x}_0 \in \tau_I$ and for each $i \in \{1, \ldots, n\}$, the pair $(\bar{x}_{i-1}, \bar{x}_i)$ lies in the region δ_{a_i}. The run ρ is *accepting* if the sequence terminates in a configuration $\bar{x}_n \in \tau_F$. The *language* recognized by the $\mathcal{D}$-automaton $\mathcal{A}$ is the set of words for which there is an accepting run. In Section 4 we will consider automata over infinite words, but right now we focus on finite words. As usual, we are mostly interested in determining emptiness of such automata.

▶ **Example 1.** We define a $\mathcal{D}$-automaton $\mathcal{A}$ of dimension 2, where $\mathcal{D} = \langle \mathbb{Q}, <, 0 \rangle$ or $\mathcal{D} = \langle \mathbb{N}, <, 0 \rangle$ and $A = \{a, b\}$. The regions τ_I, τ_F are both described by $x_1 \geq 0 \wedge x_2 \geq 0$. The region δ_b is the set of points (x_1, x_2, y_1, y_2) such that $y_2 = x_2 \wedge x_1 < y_1 < x_2$ and δ_a is specified by $y_2 = x_2 \wedge y_1 = 0$. The language recognized by $\mathcal{A}$ is $(b^*a)^*$.

▶ Remark. It is often convenient to equip $\mathcal{D}$-automata with states. This does not change the expressive power as states can be simulated using extra dimensions.

▶ Remark. Since one can construct Cartesian products of $\mathcal{D}$-automata, the languages they recognize are closed under union and intersection.

▶ Remark. We could also use existential formulas for defining the transitions of the automaton. This would not affect decidability nor expressiveness. However, if we allowed a logic which can define the successor relation (such as first-order logic, when working over the naturals), we would easily encode a Minsky machine into the model, resulting in an undecidable emptiness problem.

Results. In the next section we develop a framework for manipulating $\mathcal{D}$-automata based on the notions of *potential* and *saturation.* We illustrate the use of these notions by proving that for any linearly ordered structure $\mathcal{D}$ and $\mathcal{D}$-automaton $\mathcal{A}$, the language recognized by $\mathcal{A}$ is regular by exhibiting an equivalent finite state automaton, yielding:

▶ **Theorem 2.** *For any linearly ordered structure $\mathcal{D}$, $\mathcal{D}$-automata recognize precisely the class of regular languages.*

We will see in Section 3.4 that our proof is actually constructive assuming that a reasonable amount of computation can be performed on $\mathcal{D}$. Our construction brings a PSPACE complexity for the emptiness problem for all linearly ordered structures used in the literature.

An analogue of Theorem 2 for infinite words no longer holds. However, with further computational assumptions, we will extend the decidability of emptiness to the infinite setting in Section 4. In Section 5 we show how LTL formulas using constraints expressible over $\mathcal{D}$ can be translated into $\mathcal{D}$-automata. Finally in Section 6 we show how our framework can also be used to solve the case where an underlying finite database is present.

3 Finite words

3.1 Cell automata

In this section we fix a linearly ordered structure $\mathcal{D}$ and a $\mathcal{D}$-automaton $\mathcal{A}$ of dimension k, described by the tuple $((\delta_a)_{a \in A}, \tau_I, \tau_F)$. We construct a finite nondeterministic automaton $\mathcal{A}'$, called the *cell automaton*, targeted at simulating the runs of $\mathcal{A}$.

The states of $\mathcal{A}'$ are the cells of $\mathcal{D}^k$. The automaton $\mathcal{A}'$ has a transition from the state τ to the state τ' labeled by the input letter a if and only if there exist $\bar{x} \in \tau$ and $\bar{y} \in \tau'$ such that $(\bar{x}, \bar{y}) \in \delta_a$. The initial states of $\mathcal{A}'$ are those cells τ for which $\tau \subseteq \tau_I$. The accepting states are those cells τ for which $\tau \subseteq \tau_F$. The following proposition is rather immediate:

▶ **Proposition 3.** *The language recognized by $\mathcal{A}$ is included in the language recognized by $\mathcal{A}'$.*

The converse inclusion does not always hold, as shown in the following example.

▶ **Example 4.** Let $\mathcal{D} = \langle \mathbb{Z}, <, 0, 3\rangle$ and let $\mathcal{A}$ be the $\mathcal{D}$-automaton of dimension 1 described as follows. The region δ_a is the subset of $\mathcal{D}^2$ given by the condition $0 < x < y < 3$; the initial and final regions are defined by $0 < x < 3$. $\mathcal{A}$ accepts $\{\epsilon, a\}$. However, the cell automaton $\mathcal{A}'$ corresponding to $\mathcal{A}$ has a self-loop in the state corresponding to the cell $0 < x < 3$ and therefore recognizes the language a^*.

3.2 Potential

We can prove the correctness of the cell construction if $\mathcal{D}^k$ is equipped with a sort of inductive measure called the *potential* which tells, roughly, given a point $\bar{x} \in \mathcal{D}^k$, how long runs of the cell automaton can be lifted to runs of the original automaton, starting from the point $\bar{x}$. Formally, a function $E\colon \mathcal{D}^k \to \mathbb{N} \cup \{\infty\}$ is called a *potential* for $\mathcal{D}^k$ if it satisfies the following conditions.

1. *Cells have unbounded potential*: For any cell $\theta \subseteq \mathcal{D}^k$ and any number $s \geq 0$ there exists a point $\bar{x} \in \theta$ such that $E(\bar{x}) \geq s$.
2. *Transitions decrease the potential by at most 1*: Let $\tau, \tau' \subseteq \mathcal{D}^k$, $\delta \subseteq \mathcal{D}^{2k}$ be cells such that there exists $(\bar{x}_0, \bar{y}_0) \in \delta$ with $\bar{x}_0 \in \tau$ and $\bar{y}_0 \in \tau'$. Then, if $\bar{x} \in \tau$ is such that $E(\bar{x}) \geq s + 1$ for some $s \geq 0$, there exists a point $\bar{y} \in \tau'$ such that $(\bar{x}, \bar{y}) \in \delta$ and $E(\bar{y}) \geq s$.

▶ **Example 5.** If $\mathcal{D} = \langle \mathbb{Q}, <, 0, 3\rangle$, the mapping constantly equal to ∞ is a potential for $\mathcal{D}^n$, for any $n \in \mathbb{N}$. This follows easily from the fact that $\mathbb{Q}$ is a dense linear order.

On the other hand, if $\mathcal{D} = \langle \mathbb{Z}, <, 0, 3\rangle$ as in Example 4, then $\mathcal{D}^1$ cannot be equipped with a potential. Indeed, the cell $\tau: \ 0 < x < 3$ contains only two points, $\bar{x}_0 = 1$ and $\bar{y}_0 = 2$. By using the second axiom, it is not difficult to prove that both points in τ must have a potential smaller than 2, contradicting the first axiom of the potential.

However, if $\mathcal{D} = \langle \mathbb{Z}, <, 0, 1, 2, 3\rangle$, there is a potential for $\mathcal{D}^1$ which assigns to a point x the value ∞ if $x \in \{0, 1, 2, 3\}$ and the distance from x to the "critical set" $\{0, 1, 2, 3\}$ otherwise. Intuitively, a cell in $\mathcal{D}^2$, seen as a transition, either sets the value of the variable to some constant, or allows the variable to attain a value arbitrarily far from the critical set, or requires the variable to get closer by at least one to the critical set. In any case, taking a point $x \in \mathbb{Z}$ with potential at least s, we can follow any transition and guarantee to end up in a point with potential at least $s - 1$.

Finally, consider $\mathcal{D}^2$, where $\mathcal{D} = \langle \mathbb{Z}, <, 0, 1, 2, 3\rangle$. Then $\mathcal{D}^2$ can also be equipped with a potential which, roughly, to a given point (x, y) assigns a value which depends on how far the variables x, y are from the constants $0, 1, 2, 3$ and from each other. Such a potential is constructed in the next section.

In the presence of a potential, runs of the cell automaton induce corresponding runs of the original $\mathcal{D}$-automaton. This is phrased in the following proposition, whose proof is obtained from the definition by an easy induction.

▶ **Proposition 6.** *Let $\mathcal{D}^k$ be equipped with a potential E, let $\mathcal{A}$ be a $\mathcal{D}$-automaton of dimension k and let $\mathcal{A}'$ be the cell automaton corresponding to $\mathcal{A}$. Let $\tau_0 \to \tau_1 \to \cdots \to \tau_n$ be a run of the cell automaton over a word $a_1 a_2 \ldots a_n$. Then, for all numbers $s \geq 0$ and for all points $\bar{x}_0 \in \tau_0$ such that $E(\bar{x}_0) \geq s + n$ there exists a sequence of points $\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n \in \mathcal{D}^k$ such that $\bar{x}_i \in \tau_i$, $E(\bar{x}_i) \geq s + i$ and $(x_{i-1}, x_i) \in \delta_{a_i}$ for all $i = 1, \ldots n$.*

In the above proposition, accepting runs of $\mathcal{A}'$ clearly induce accepting runs of $\mathcal{A}$. Therefore, we immediately obtain the following.

▶ **Theorem 7.** *Assume that there is a potential E for $\mathcal{D}^k$. Then, for any $\mathcal{D}$-automaton of dimension k, the corresponding cell automaton $\mathcal{A}'$ recognizes the same language as $\mathcal{A}$.*

3.3 Constructing the potential and saturation

When $\mathcal{D}$ cannot be equipped with a potential, we show how to extend $\mathcal{D}$ to a structure which is *saturated.* For such structures, we are always able to define a potential. Altogether, this will prove Theorem 2. In Example 4 the saturation process will add the constants $1, 2$ to the structure, allowing the new cell automaton to count up to 3.

Let $\mathcal{D}$ be a linear order. A *virtual element* of $\mathcal{D}$ is a subset S of D which is downward closed (i.e. $x \in S \wedge y \leq x \implies y \in S$), but is not of the form $\{y \mid y < x\}$ or $\{y \mid y \leq x\}$ for some $x \in \mathcal{D}$. For an element $x \in D$ and a virtual element S, we will write $x < S$ if $x \in S$, and $x > S$ otherwise. Also, for two virtual elements S, S' we can write $S < S'$ if $S \subsetneq S'$. We denote by $\bar{D}$ the set of elements and virtual elements of D, linearly ordered by $<$ (known as the Dedekind completion, modulo the elements $\emptyset, D$ which are normally not considered in $\bar{D}$). Note that $\bar{D}$ has a smallest and largest element, denoted $c_{-\infty}$ and c_{∞}, respectively.

In a linearly ordered structure $\mathcal{D}$ we distinguish between two kinds of unary predicates. We denote those that correspond to virtual elements, as they play a crucial role in our proofs, as *virtual constant* and we treat them as constants. In particular we will use symbols c, d for both virtual and real constants. As an example, consider the linearly ordered structure $\langle \mathbb{Q}, <, e \rangle$ where e is a virtual constant corresponding to the set $\{x \in \mathbb{Q} \mid x < 2.718\ldots\}$.

For a point $x \in D$, we define its *type*, denoted $t(x)$ as the set of unary predicates (including virtual constants) it satisfies and constants which it is equal to.

To simplify the following definitions, we will assume that $c_{-\infty}$ and c_{∞} are (possibly virtual) constants of $\mathcal{D}$. Let $w = t_1, \ldots, t_n$ be a sequence of types. We say that a sequence of points $x_1 < \ldots < x_n$ *realizes* w if $t(x_i) = t_i$ for all i. For an interval I, we say that w is *realizable in* I if some sequence $x_1 < \ldots < x_n$ of elements of I realizes w.

We construct a function, called *quasi-potential*, $qE \colon \mathcal{D}^* \to \mathbb{N} \cup \{\infty\}$, defined on all tuples of elements of $\mathcal{D}$. It will give raise to a potential defined on $\mathcal{D}^k$, for all k. Intuitively, $\bar{x}$ has high potential if long sequences of types can be realized in between any two coordinates of $\bar{x}$ or between a coordinate of $\bar{x}$ and a constant. The precise definition is given below.

Let $\bar{x}$ be a point in $\mathcal{D}^k$. Let $\{u_1, u_2, \ldots, u_s\}$ be the union of the set of coordinates of $\bar{x}$ and of the set of constants and virtual constants of $\mathcal{D}$. We assume that $u_1 < u_2 < \ldots < u_s$. For $x \in \bar{D}$ let $\underline{c}(x)$ be the largest (virtual) constant c such that $c \leq x$ and $\overline{c}(x)$ be the smallest (virtual) constant c such that $x \leq c$. For $1 \leq i < s$, let T_i be the set of types occurring in $]\underline{c}(u_i), \overline{c}(u_{i+1})[$. For each $0 \leq i < s$, the *capacity* of the interval $]u_i, u_{i+1}[$ is the length of the shortest sequence of elements of T_i which is not realizable in $]u_i, u_{i+1}[$ or as ∞ if all such sequences are realizable. Finally, we define

$$qE(\bar{x}) = \min\{\text{capacity of }]u_i, u_{i+1}[\mid 1 \leq i < s\}.$$

In the case where $k = 0$, $\mathcal{D}^0$ contains only one element – the empty tuple ε. We say that $\mathcal{D}$ is *saturated* if $qE(\varepsilon) = \infty$. In other words, for any two consecutive (virtual) constants c, d, any sequence of types which occur between c and d is realizable between c and d. Any linearly ordered structure can be transformed into a saturated one:

▶ **Theorem 8.** *Let $\mathcal{D}$ be a linearly ordered structure. It is possible to expand $\mathcal{D}$ with a finite set of constants and virtual constants to obtain a saturated structure $\hat{\mathcal{D}}$.*

Proof. We say that a linear order $\mathcal{D}$ is *complete* if any subset of D has its supremum, i.e. a least upper bound (again, this deviates slightly from the standard notion of completeness by the least and smallest elements). Examples of complete linear orders are $\mathbb{N} \cup \{\infty\}, \mathbb{R} \cup \{-\infty, +\infty\}$ but not $\mathbb{Q} \cup \{-\infty, +\infty\}$ nor $\mathbb{R}$. The following lemma solves the case of complete linear orders.

▶ **Lemma 9.** *Let $\mathcal{D}$ be a complete linear order. Let $a < b$ be two points in D and let $t\colon]a, b[\to T$ be a function with $|T| = n < \infty$. Then there exist $a = d_0 < d_1 < \ldots < d_m = b$ such that for any interval $I =]d_i, d_{i+1}[$ and $w \in (t(I))^*$, the sequence w is realizable in I.*

Proof. We prove the claim by induction on n. If $n = |T| = 0$ there is nothing to prove. Let us assume that we have proved the proposition for all $|T| < n$. Let s denote the length of the shortest sequence $w \in T^*$ which is not realizable in the interval $]a, b[$. We do a nested induction on s. If $s = 1$, there is a $u \in T$ which does not occur in $]a, b[$, so we may reduce the set T to $T \setminus \{u\}$ with less than n elements. Let us assume that $s \geq 2$.

Let $w = t_1, \ldots, t_s \in T^*$ be the sequence of length s which is not realizable in the interval $]a, b[$. We will define an element c such that for both open intervals $]a, c[$ and $]c, b[$, there is a non-realizable sequence of length smaller than s.

If $s > 2$ then the sequence t_1, t_s is realizable in $]a, b[$, so let $x < y$ realize it. Let $c = x$. Then the sequence $t_1, t_2, \ldots, t_{s-1}$ is not realizable in the interval $]a, c[$ and the sequence $t_2, t_3, \ldots, t_s$ is not realizable in the interval $]c, b[$.

If $s = 2$, the sequence t_1, t_2 is not realizable in $]a, b[$. Let us consider the supremum c of all possible x with $t(x) = t_2$. Then, t_2 is not realizable in the interval $]c, b[$. Moreover, t_1 is not realizable in the interval $]a, c[$ — otherwise, t_1, t_2 would be realizable in $]a, b[$.

We apply the inductive assumption to $]a, c[$ and $]c, b[$ obtaining sequences $a = d_0 < d_1 < \ldots < d_m = c$ and $c = d_m < d_{m+1} \ldots < d_{m'} = b$ such that for any interval $I =]d_i, d_{i+1}[$ and $w \in (t(I))^*$, the sequence w is realizable in I. This ends the inductive proof of the lemma. ◀

Let $\mathcal{D} = \langle D, <, P_1, P_2, \ldots, P_l, c_1, c_2, \ldots, c_m \rangle$ be an arbitrary linearly ordered structure. It is well known that $\langle \bar{D}, < \rangle$ is a complete linear order. Let us consider the structure

$$\bar{\mathcal{D}} = \langle \bar{D}, <, P_0, P_1, P_2, \ldots, P_l, c_1, c_2, \ldots, c_m \rangle,$$

where P_0 is the unary predicate corresponding to the set $D \subseteq \bar{D}$. Let $t\colon \bar{D} \to T$ be the function which assigns to an element of $\bar{\mathcal{D}}$ its type, i.e. the set of unary predicates it satisfies and constants equal to it. We apply Lemma 9 to $\bar{\mathcal{D}}$ and the points $c_{-\infty} < c_{\infty}$. We obtain a sequence of elements $d_0 < d_1 < \ldots < d_{m'}$ such that for any interval $I =]d_i, d_{i+1}[$ and $w \in (t(I))^*$, the sequence w is realizable in I. We define $\hat{\mathcal{D}}$ as the extension of $\mathcal{D}$ by the (possibly virtual) elements $d_0, \ldots, d_{m'}$ as constants. It is easy to verify that $\hat{\mathcal{D}}$ is saturated. ◀

▶ **Example 10.** We apply the procedure of Theorem 8 to $\mathcal{D} = \langle \mathbb{Z}, <, 0, 3 \rangle$ of Example 4. The only relevant interval is $]0, 3[$. There is only one type t in $]0, 3[$ and ttt is not realized. Since tt is realized by $1 < 2$, we are in the first case and the constant 1 is added to $\mathcal{D}$. In the next step, the relevant interval is $]1, 3[$ and tt is not realized in it. Adding the supremum of all elements of type t in $]1, 3[$, i.e. the constant 2, yields a saturated structure.

To see why we might need to also add a virtual constant, consider the linearly ordered structure $\mathcal{D} = \langle \mathbb{Q}, <, 0, 3, P, Q \rangle$ where $P = \{(1-1/n)^n | n \in \mathbb{N}\}$ and $Q = \{(1+1/n)^{-n} | n \in \mathbb{N}\}$. Consider the interval $]0, 3[$. As $P \cap Q = \emptyset$ there are three types occurring in this interval: $t_1 = \{P\}$, $t_2 = \{Q\}$, $t_3 = \emptyset$. Since $t_1 t_2$ is not realized in $]0, 3[$, the procedure of Theorem 8 introduces the supremum of all elements of type t_2 in this interval, i.e. the virtual constant e. The reader can verify that the resulting structure $\hat{\mathcal{D}} = \langle \mathbb{Q}, <, 0, 3, e, P, Q \rangle$ is saturated.

▶ **Theorem 11.** *If $\mathcal{D}$ is a saturated linearly ordered structure and $k \geq 0$ then there is a potential E for $\mathcal{D}^k$, given by $E(\bar{x}) = \lfloor \log_{3^k}(qE(\bar{x})) \rfloor$.*

Proof sketch. First we show that if $\bar{x} \in \mathcal{D}^k$ has $qE(\bar{x}) \geq s$ for some large s, then we can extend it to a point $(\bar{x}, y_1)$ laying in a prescribed cell τ_1 in $\mathcal{D}^{k+1}$, and with $qE(\bar{x}, y_1) \geq s'$ for some still large s'. We need to assume that τ_1 is a "reasonable" cell – i.e. the projection of τ_1 onto the first k coordinates contains the point $\bar{x}$. The cell then τ_1 typically requires that y_1 lays in the interval between some two coordinates of $\bar{x}$, and moreover is of some type t. The idea is to choose y_1 to lie "in the middle" of the prescribed interval, obtaining a point with quasi-potential larger than a third of s. Iterating this process, we insert k coordinates, getting a point $(\bar{x}, \bar{y})$ which lays in a prescribed cell τ in $\mathcal{D}^{2k}$ with $qE(\bar{x}, \bar{y}) \geq \frac{s}{3^k}$. We conclude that $qE(\bar{y}) \geq \frac{s}{3^k}$. By taking the appropriate logarithm, we construct the actual potential. To show that qE is unbounded on all cells, we reuse the above reasoning: we start with the empty tuple ε with $qE(\varepsilon) \geq s$ for all s by saturation, and insert coordinates one after the other, preserving high qE-value, and finally obtaining a point in the desired cell. ◀

3.4 Computability and Complexity issues

We now turn to the complexity analysis. In practical applications, a base linearly ordered structure $\mathcal{D}$ is fixed and new constants come with the description of the automaton. Here, and in other sections, for a finite set $C \subseteq \mathcal{D}$, we denote by $\mathcal{D}[C]$ the structure $\mathcal{D}$ extended by the constants in C. The above motivation leads to the following decision problem, which we call *emptiness of $\mathcal{D}[C]$-automata.* **Input:** 1) A set C of elements of $\mathcal{D}$, encoded in some specified presentation 2) A description of a $\mathcal{D}[C]$-automaton $\mathcal{A}$. **Decide:** is $\mathcal{A}$ empty?

Our decision algorithm essentially works as follows: We first compute a saturated expansion $\hat{\mathcal{D}}$ of the structure $\mathcal{D}[C]$. We then compute in $\hat{\mathcal{D}}$ the cell automaton $\mathcal{A}'$ associated to $\mathcal{A}$ and check its emptiness. The correctness of this algorithm follows from the results of the previous sections. In order to decrease the complexity, we materialize neither $\hat{\mathcal{D}}$ nor $\mathcal{A}'$ but compute them on the fly. For $\hat{\mathcal{D}}$ we need to know the new (virtual) constants that were introduced together with their type. For $\mathcal{A}'$ we essentially need to know the possible types that may occur in the interval between any two successive (virtual) constants of $\hat{\mathcal{D}}$. We therefore need to assume that $\mathcal{D}$ is equipped with an algorithm giving us this information. This is formalized as follows.

A sequence $s : t_0 T_1 t_1 T_2 \cdots t_{n-1} T_n t_n$ where $t_0, t_1, \ldots, t_{n-1}, t_n$ are types and $T_1, T_2, \ldots, T_n$ are sets of types of $\mathcal{D}$ is called a *saturator.* Given two elements x and y of $\mathcal{D}$, we say that a sequence $c_0 < c_1 < \ldots < c_n$ of (possibly virtual) elements of $\mathcal{D}$ *matches s in $[x, y]$* if $c_0 = x$, $c_n = y$, c_i is of type t_i and T_i^* are precisely the sequences of types realizable in $]c_{i-1}, c_i[$.

▶ **Example 12.** Over $\langle \mathbb{Z}, <, 0 \rangle$, the saturator $t_\emptyset T_\epsilon t_0 T_\epsilon t_\emptyset T_\epsilon t_\emptyset$, where $t_\emptyset$ is the empty type, t_0 is the type of 0 and $T_\epsilon = \{\}$, is matched in $[-1, 2]$ by the sequence $-1 < 0 < 1 < 2$.

▶ **Example 13.** Over $\langle \mathbb{Q}, <, 0, P \rangle$, where $P = \{(1 + 1/n)^n \mid n \in \mathbb{N}^+\}$, the sequence $-100 < 0 < 2 < e < 100$ matches the saturator $t_\emptyset T_\emptyset t_0 T_\emptyset t_P T_P t_\emptyset T_\emptyset t_\emptyset$ in $[-100, 100]$ where t_P is the type P, $T_P = \{t_\emptyset, t_P\}$, $T_\emptyset = \{t_\emptyset\}$, while t_0 and $t_\emptyset$ are as before.

A saturator is *realizable* in $[x, y]$ if there is a sequence which matches it in $[x, y]$. A linearly ordered structure $\mathcal{D}$ is said to be *computable* if there is an algorithm that given any two elements x, y of $\mathcal{D}$ replies the length of a saturator s realizable in $[x, y]$, and afterwards, when given a number i as input, returns the i^{th} element of the sequence s. When this algorithm works in PTime, we say that $\mathcal{D}$ is P-computable. Note that for integers or rationals, an obvious saturator algorithm runs in PTIME, even if the numbers are coded in binary. By a careful analysis of the proof of Theorem 2 we obtain the following.

▶ **Theorem 14.** *If $\mathcal{D}$ is* P*-computable (respectively, computable) the emptiness problem of $\mathcal{D}[C]$-automata is in* PSPACE *(respectively, decidable).*

Sketch. Assume $\mathcal{D}$ is P-computable. Let $\mathcal{A}$ be a $\mathcal{D}[C]$-automaton. For each pair of consecutive elements in C, we invoke the saturator algorithm. Let l be maximal of the lengths of the saturators as returned by the saturator algorithm. Since l has polynomial size, each (virtual) constant in the saturated expansion $\hat{\mathcal{D}}$ of $\mathcal{D}[C]$ for that interval has a polynomial size presentation. Hence, in the end, each cell of $\hat{\mathcal{D}}$ has a polynomial size representation. Given two cells, it can be checked in polynomial time whether there is a transition in the cell automaton associated to $\mathcal{A}$. This is because $\hat{\mathcal{D}}$ is saturated, so only local consistency needs to be checked – namely, whether the type associated with each variable is indeed a possible type in the interval where this variable must be realized – and this information is also provided by the saturator algorithm. Decidability then follows by guessing on the fly the appropriate sequence of cells. ◀

4 Infinite words

We now consider infinite words. Recall that a *$\mathcal{D}$-automaton $\mathcal{A}$ of dimension k* consists of the transition regions $(\delta_a)_{a\in A}$, an initial region $\tau_I \subseteq \mathcal{D}^k$ and an accepting region $\tau_F \subseteq \mathcal{D}^k$. A *Büchi $\mathcal{D}$-automaton* is an automaton over infinite words, in which a run ρ is declared *accepting* if it visits infinitely often the region[2] τ_F.

The following example shows that Theorem 2 does not directly extend to infinite words.

▶ **Example 15.** An infinite sequence of numbers $n_1, n_2, \ldots$ induces an infinite word $b^{n_1}ab^{n_2}\ldots$ Let $\mathcal{L}$ be the language of words which are induced by bounded sequences. Then $\mathcal{L}$ is recognized by the Büchi $\mathcal{D}$-automaton $\mathcal{A}$ described in Example 1, where $\mathcal{D} = \langle \mathbb{N}, <, 0\rangle$.

The language $(b^*a)^\omega \setminus \mathcal{L}$ does not contain any ultimately periodic word. In particular, $\mathcal{L}$ cannot be ω-regular. We will see that also nonempty Büchi $\mathcal{D}$-automata must accept some ultimately periodic word, so we deduce that they are not closed under complementation.

Although Büchi $\mathcal{D}$-automata are more expressive than Büchi automata, we show that emptiness can be reduced to the finite case, under additional computability assumptions concerning $\mathcal{D}$. We need the following notions. For a type t, we say that a (virtual) element x of $\mathcal{D}$ is a *left t-limit* if for every $y \in D$ such that $y < x$, the type t occurs between y and x. A *right t-limit* is defined dually. The *ω-type* of a point $x \in \bar{D}$ is its type extended by the specification, for all types t, whether it is a left or right t-limit or none. An *ω-saturator* is a sequence of the form $t_0T_1t_1T_2\cdots T_nt_n$ where each t_i is an ω-type and each T_i is a set of ω-types. We extend the notion of *matching* to the case of ω-saturators in an obvious way. Next, we define (P-)*ω-computability* analogously to (P-)computability, where, for given $x, y \in D$ the algorithm should calculate an ω-saturator realizable in $[x, y]$ treated as a subset of $\bar{D}$. Note that the structures considered earlier are P-ω-computable.

▶ **Theorem 16.** *For a* P*-ω-computable (resp. ω-computable) linearly ordered structure $\mathcal{D}$, emptiness of Büchi $\mathcal{D}[C]$-automata is in* PSPACE *(resp. decidable).*

[2] An acceptance condition requiring that ρ visits infinitely often a point $\bar{x} \in \tau_F$ yields a weaker model.

It appears that $\mathcal{D}$-automata are related with ωB-automata, a model defined in [2]. Informally, an ωB-*automaton* is a nondeterministic automaton equipped with counters which can be incremented and reset, but not tested during the run. The acceptance condition of the automaton, apart from a Büchi condition, requires that the counters remain bounded when processing the infinite input word. We call a language recognized by a $\mathcal{D}$-automaton (resp. ωB-automaton) a $\mathcal{D}$-*regular* (resp. ωB-*regular*) language.

▶ **Theorem 17.** *If $\mathcal{D} = \langle \mathbb{Q}, <, c_1, c_2, \dots, c_m \rangle$ then the classes of $\mathcal{D}$-regular languages and ω-regular languages coincide. If $\mathcal{D} = \langle \mathbb{N}, <, c_1, c_2, \dots, c_m \rangle$ then the classes of $\mathcal{D}$-regular languages and ωB-regular languages coincide. Moreover, all translations are effective.*

It appears that the above dichotomy is valid for all linearly ordered structures $\mathcal{D}$, depending on whether a discrete set is definable in $\mathcal{D}$ (this can be formalized). The general result will appear in the journal version of this paper.

As a conclusion from the strong complementation result of [2], we obtain:

▶ **Corollary 18.** *Let A be a finite alphabet and $\mathcal{D}$ be as in the above theorem. It is decidable whether a Boolean combination of languages accepted by $\mathcal{D}$-automata over A is empty.*

5 Temporal logic

We fix a linearly ordered structure $\mathcal{D}$. We consider a variant of LTL where each atomic predicate is replaced by a proposition comparing the current configuration with the next one. We denote this logic by LTL($\mathcal{D}$). Its syntax is given by the following grammar:

$$\begin{aligned} \varphi &:: \ \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \varphi \mid \psi \\ \psi &:: \ \psi \wedge \psi \mid \psi \vee \psi \mid \neg\psi \mid a \mid P(\alpha) \mid \alpha = \alpha \mid \alpha < \alpha \\ \alpha &:: \ x \mid \mathsf{X}x \mid c \end{aligned}$$

where $x \in \{x_1, x_2, \dots\}$ are variables, $a \in A$ letters, $c \in \mathcal{D}$, and P unary predicates of $\mathcal{D}$.

The semantics of a formula $\varphi \in$ LTL($\mathcal{D}$) is defined on sequences of elements in $A \times D^k$, where k is the maximal number such that x_k appears in φ. Let $w = (a_1, \bar{z}_1)(a_2, \bar{z}_2)\dots$ be such a sequence. Let ψ be a proposition as described by the grammar above and let n be a position of w. We write $(w, n) \models \psi$ if from ψ we obtain a sentence which is valid in $\mathcal{D}$ after replacing the terms of the form x_i by the i^{th} coordinate of $\bar{z}_n$, and terms of the form $\mathsf{X}x_i$ by the i^{th} coordinate of $\bar{z}_{n+1}$. Using the classical semantics of LTL, we extend this notation to all formulas φ of LTL($\mathcal{D}$), and say that w is *accepted* by φ if $(w, 1) \models \varphi$.

The following result can be established along the same lines as in the classical translation of LTL formulas into automata. C_φ denotes the set of values which appear in the formula φ.

▶ **Theorem 19.** *For any LTL($\mathcal{D}$) formula φ there exists a $\mathcal{D}[C_\varphi]$-automaton $\mathcal{A}_\varphi$ whose runs are exactly the sequences accepted by φ.*

▶ Remark. In [8] terms of the form $\mathsf{X}^j x$ were allowed, enabling to compare the data value with one that will occur j steps later. This can be simulated by a formula of LTL($\mathcal{D}$) after adding new dimensions used to guess in advance the values that will appear in the following j steps. The consistency of the guesses can be enforced at each step by a formula of LTL($\mathcal{D}$).

▶ Remark. It is tempting to consider other temporal formalisms. One could define a variant of μ-calculus analogously to the extension LTL($\mathcal{D}$) described above, and our model of automata still can simulate such formulas. However, as noticed by [7, 11, 8] over various domains, CTL($\mathcal{D}$) is undecidable.

6 Databases

Following [11] we apply in this section the results of the previous sections in the context where a finite database is present. For this, we fix a relational schema σ and a linearly ordered structure $\mathcal{D}$. A database over σ is then, for each relation symbol of σ, a finite relation of the appropriate arity over the domain of $\mathcal{D}$.

A $(\mathcal{D},\sigma)$*-automaton* $\mathcal{A}$ *of dimension* k is described as follows. As before, the configurations of $\mathcal{A}$ are points in the space $\mathcal{D}^k$. We assume an initial region $\tau_I \subseteq \mathcal{D}^k$ and an accepting region $\tau_F \subseteq \mathcal{D}^k$. For each $a \in A$, the transition δ_a is a set of pairs of the form (τ,φ), where τ is a region in $\mathcal{D}^k \times \mathcal{D}^k$, while φ is a propositional formula over σ with $2k$ free variables. Given a finite database M over the schema σ and two points $\bar{x},\bar{y} \in \mathcal{D}^k$, we write $\bar{x} \xrightarrow{a}_M \bar{y}$ iff there is a pair $(\tau,\varphi) \in \delta_a$ with $(\bar{x},\bar{y}) \in \tau$ and $M \models \varphi(\bar{x},\bar{y})$. A *run* ρ *of* $\mathcal{A}$ *on* $w = a_1a_2\dots$ *over* M, is a sequence of configurations $\bar{x}_0,\bar{x}_1,\ldots \in \mathcal{D}^k$ such that $\bar{x}_0 \in \tau_I$ and for each $n > 0$, $\bar{x}_{n-1} \xrightarrow{a_n}_M \bar{x}_n$. Acceptance conditions are defined as before, for finite or infinite runs.

▶ **Example 20.** We fix $\mathcal{D} = \langle \mathbb{Q},<,0\rangle$ and $\sigma = \{P\}$ where P is unary. Consider the $(\mathcal{D},\sigma)$-automaton $\mathcal{A}$ of dimension 1, where τ_I,τ_F are both described by $x = 0$. The region τ_b is the set of points (x,y) such that $x < y$ and δ_b is the pair $(\tau_b, P(y))$ while δ_a is just specified by the region $y = 0$. Hence, the length of any sequence of b's is bounded by the size of the database, so the infinite words w for which there exists a database M and a run of $\mathcal{A}$ consistent with w and M are exactly the bounded sequences of Example 15. Recall from Theorem 17 that without the underlying database $\mathcal{D}$-automata would only recognize ω-regular languages.

Our goal is to decide whether, for a given $(\mathcal{D},\sigma)$-automaton $\mathcal{A}$, there exists a finite database M such that A has an accepting run over M.

▶ Remark. A more general setting would allow the database constraints φ to be existential queries with $2k$ free variables. This setting can be easily reduced to the one above, by having the automaton $\mathcal{A}$ guess the values for the quantified variables using extra dimensions.

It appears that adding the database does not influence the decidability results obtained in Theorem 14 and Theorem 16 for finite and infinite words, respectively. We state the result in the database setting.

▶ **Theorem 21.** *Let $\mathcal{D}$ be a computable (resp. ω-computable) linearly ordered structure. Given a $(\mathcal{D}[C],\sigma)$-automaton $\mathcal{A}$, it is decidable whether there exists a finite database M and a finite (resp. infinite) word w such that there is an accepting run of $\mathcal{A}$ on w over M. Moreover if $\mathcal{D}$ is* P*-computable (resp.* P*-ω-computable) then the complexity is* PSPACE *for a fixed schema,* EXPSPACE *otherwise.*

Sketch. We only sketch here the ideas for the case of finite words. The case of infinite runs is done by a reduction to the finite one in a way similar to the proof of Theorem 16. We temporarily treat $\mathcal{A}$ as a $\mathcal{D}[C]$-automaton, for each a merging into one all regions appearing in the transition δ_a. Let $\mathcal{A}'$ be the corresponding cell automaton, over the saturated expansion $\hat{\mathcal{D}}$ of $\mathcal{D}[C]$. A run π' of $\mathcal{A}'$ is lifted to a run π of $\mathcal{A}$ inductively, assuring that in each step we obtain a configuration with a big potential. The key observation is that the new configuration can be chosen so that it does not use values which appeared in previous configurations, unless it is explicitly required by the transition. Hence it is enough to guess the database relations for the constants $\hat{\mathcal{D}}$ (this is exponential in the maximal arity of a relation in σ, hence the complexity becomes EXPSPACE unless this arity is fixed) and maintain locally, by adding states to $\mathcal{A}'$, the consistency of the database. Each time a new configuration reuses some data values, this is enforced by the transition and hence consistency can be tested by $\mathcal{A}'$ at the time of the transition. Otherwise, as we observed, fresh data values can always be chosen and consistency with the previous constraints is immediate. ◀

Logic. The logic LTL($\mathcal{D}$) described in Section 5 can be extended to a logic LTL($\mathcal{D}, \sigma$) by adding, for each symbol E in σ of arity k, a production $\psi :: E(\alpha, \alpha, \dots, \alpha)$, in which the right-hand side has k arguments. Atoms of the form $E(x, \mathsf{X}y)$, $F(x, y, \mathsf{X}x)$ etc. represent database queries. Just as before, the logic LTL($\mathcal{D}, \sigma$) can be transformed into $(\mathcal{D}, \sigma)$-automata, and thus can be effectively verified. We omit the details in this abstract.

7 Conclusions

We have introduced an automata model capable of storing values from a linearly ordered set, additionally equipped with constants and unary predicates. We have shown how to simulate runs of such automata by finite state automata. This translation is effective as soon as the structure has some reasonable computational properties. This provides a uniform presentation for results concerning specific data domains that were disseminated in various papers. Moreover this sometimes decreases the known complexity or solves open questions.

It would be interesting to see whether our work can be extended to other structures. We leave this for future work.

References

1 R. Alur, P. Cerný, and S. Weinstein. Algorithmic analysis of array-accessing programs. In *Computer Science Logic (CSL)*, pages 86–101, 2009.

2 M. Bojańczyk and T. Colcombet. Bounds in ω-regularity. In *Logic in Computer Science (LICS)*, pages 285–296, 2006.

3 M. Bojańczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *Logic in Computer Science (LICS)*, pages 7–16, 2006.

4 A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *Fundamentals of Computation Theory (FCT)*, pages 1–22, 2007.

5 A. Bouajjani, Y. Jurski, and M. Sighireanu. A generic framework for reasoning about dynamic networks of infinite-state processes. In *TACAS'07*, 2007.

6 P. Bouyer, A. Petit, and D. Thérien. An algebraic approach to data languages and timed languages. *Information and Computation*, 182(2), 2003.

7 K. Čerāns. Deciding properties of integral relational automata. In *ICALP*, pages 35–46, 1994.

8 S. Demri. Linear-time temporal logics with Presburger constraints: An overview. *J. of Applied Non-Classical Logics*, 16(3-4):311–347, 2006.

9 S. Demri and R. Gascon. The effects of bounding syntactic resources on Presburger LTL. *Journal of Logic and Computation*, 19(6):1541–1575, 2009.

10 S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.

11 A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *Intl. Conf. on Database Theory (ICDT)*, pages 252–267, 2009.

12 X. Du, C. R. Ramakrishnan, and S. A. Smolka. Region graphs for infinite-state systems. unpublished manuscript, 2007.

13 D. Figueira, P. Hofman, and S. Lasota. Relating timed and register automata. In *Intl. Workshop on Expressiveness in Concurrency (EXPRESS'10)*, 2010.

14 M. Kaminski and N. Francez. Finite memory automata. *Theor. Comp. Sci.*, 134(2):329–363, 1994.

Bottom-up automata on data trees and vertical XPath*

Diego Figueira and Luc Segoufin

INRIA and ENS Cachan, LSV

Abstract

A data tree is a tree whose every node carries a label from a finite alphabet and a datum from some infinite domain. We introduce a new model of automata over unranked data trees with a decidable emptiness problem. It is essentially a bottom-up alternating automaton with one register, enriched with epsilon-transitions that perform tests on the data values of the subtree. We show that it captures the expressive power of the vertical fragment of XPath —containing the child, descendant, parent and ancestor axes— obtaining thus a decision procedure for its satisfiability problem.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Decidability, XPath, Data trees, Bottom-up tree automata

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.93

1 Introduction

We study formalisms for data trees. A data tree is a tree where each position carries a label from a finite alphabet and a *datum* from some infinite domain. This structure has been considered in the realm of semistructured data, timed automata, program verification, and generally in systems manipulating data values. Finding decidable logics or automaton models over data trees is an important quest when studying data-driven systems.

A data tree can model an XML document. One wants to decide, for example, if two properties of XML documents expressed in some formalism are equivalent. This problem is usually equivalent to the satisfiability problem. One such formalism to express properties of XML documents is the logic XPath. Although satisfiability of XPath in the presence of data values is undecidable, there are some known decidable data-aware fragments [4, 5, 1, 3]. Here, we investigate a rather big fragment that nonetheless is decidable. *Vertical*-XPath is the fragment that contains all downward and upward axes, but no *horizontal* axis is allowed.

We introduce a novel automaton model that captures vertical-XPath. We show that the automaton has a decidable emptiness problem and therefore that the satisfiability problem of vertical-XPath is decidable. The **B**ottom-**U**p **D**ata **A**utomata (or BUDA) are bottom-up alternating tree automata with one register to store and compare data values. Further, these automata can compare the data value currently stored in the register with the data value of a descendant node, reached by a downward path satisfying a given regular property. Hence, in some sense, it has a two-way behavior. However, they cannot test horizontal properties on the siblings of the tree, like “the root has exactly three children”.

Our main technical result shows the decidability of the emptiness problem of this automaton model. We show this through a reduction to the the coverability problem of a well-structured transition system (WSTS [8]). Each BUDA automaton is associated with a

* This research was funded by the ERC research project FoX under grant agreement FP7-ICT-233599.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 93–104
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

transition system, in such a way that a derivation in this transition system corresponds to a run of the automaton, and vice-versa. The domain of the transition system consists in the *abstract configurations* of the automaton, which contains all the information necessary to preserve from a (partial) bottom-up run of the automaton in a subtree in order to continue the simulation of the run from there. On the one hand we show that BUDA can be simulated using an appropriate transition relation on sets of abstract configurations. On the other hand, we exhibit a well-quasi-order (wqo) on those abstract configurations and show that the transition relation is "monotone" relative to this wqo. This makes the coverability problem (and hence the emptiness problem) decidable [8].

In terms of expressive power, we show that BUDA can express any node expression of the vertical fragment of XPath. Core-XPath (term coined in [10]) is the fragment of XPath 1.0 that captures its navigational behavior, but cannot express any property involving data. It is easily shown to be decidable. The extension of this language with the possibility to make equality and inequality tests between data values is named Core-Data-XPath in [3], and it has an undecidable satisfiability problem [9]. By "vertical XPath" we denote the fragment of Core-Data-XPath that can only use the downward axes CHILD and DESCENDANT and the upward axes PARENT and ANCESTOR (no navigation among siblings is allowed). It follows that vertical XPath is decidable, settling an open question [2, Question 5.10].

Related work. A model of *top-down* tree automata with one register and alternating control (ATRA) is introduced in [12], where the decidability of its emptiness problem is proved. ATRA are used to show the decidability of temporal logics extended with a "freeze" operator. This model of automata was extended in [5] with the name ATRA(guess, spread) in order to prove the decidability of the *forward* fragment of XPath, allowing only axes navigating downward or rightward (NEXT-SIBLING and FOLLOWING-SIBLING). The two models of automata are incomparable: ATRA can express all regular tree languages while BUDA can express unary inclusion dependency properties (like "the data values labeled by a is a subset of those labeled by b"). In order to capture vertical XPath, the switch from top-down to bottom-up seems necessary to express formulas with upward navigation, and this also makes the decidability of the emptiness problem significantly more difficult. In [5], the decidability of the forward fragment of XPath is also obtained using a WSTS. This WSTS relies on a wqo over configurations. As our automaton model is bottom-up we have to work with *sets* of configurations and had to invoke the theory of *ω^2-quasi-orderings* [11, 13] in order to derive our wqo. The paper [2] contains a comprehensive survey of the known decidability results for various fragments of XPath, most of which cannot access data values. In the presence of data values, the notable new results since the publication of [2] are the downward [4] and the forward [5] fragments, as well as the fragment containing only the successor axis [3] (the latter closely related to first-order logic with two variables). As already mentioned, this paper solves one of the remaining open problems of [2].

Organization. In Section 3 we introduce the BUDA model and we show that it captures vertical XPath. The associated well-structured transition system and the outline of the proof to show the decidability of its reachability is in Section 4. Due to space limitations many proofs are omitted or only sketched. We refer the reader to [6, Chapter 7] for detailed proofs.

2 Preliminaries

Basic notation. Let $\wp(S)$ denote the set of subsets of S, and $\wp_{<\infty}(S)$ be the set of *finite* subsets of S. Let $\mathbb{N} = \{0, 1, 2, \dots\}$, $\mathbb{N}_+ = \{1, 2, 3, \dots\}$, and let $[n] := \{1, \dots, n\}$ for any $n \in \mathbb{N}_+$. We fix once and for all $\mathbb{D}$ to be any infinite domain of data values; for simplicity in our examples we will consider $\mathbb{D} = \mathbb{N}$. In general we use letters $\mathbb{A}$, $\mathbb{B}$ for finite alphabets, the letter $\mathbb{D}$ for an infinite alphabet and the letters $\mathbb{E}$ and $\mathbb{F}$ for any kind of alphabet. By $\mathbb{E}^*$ we denote the set of finite sequences over $\mathbb{E}$, by $\mathbb{E}^+$ the set of finite sequences with at least one element over $\mathbb{E}$, and by $\mathbb{E}^\omega$ the set of infinite sequences over $\mathbb{E}$. We write ϵ for the empty sequence and '$\cdot$' as the concatenation operator between sequences. We write $|S|$ to denote the length of S (if S is a finite sequence), or its cardinality (if S is a set).

Regular languages. We make use of the many characterizations of regular languages over a finite alphabet $\mathbb{A}$. In particular, we use that a word language $\mathscr{L} \subseteq \mathbb{A}^*$ is regular iff there is a finite semigroup $(S, \cdot)$ with a distinguished subset $T \subseteq S$, and a semigroup homomorphism $h : \mathbb{A}^* \to S$ such that for all w with $|w| > 0$, $w \in \mathscr{L}$ iff $h(w) \in T$.

Unranked finite trees. By *Trees*$(\mathbb{E})$ we denote the set of finite ordered and unranked trees over an alphabet $\mathbb{E}$. We view each **position** in a tree as an element of $(\mathbb{N}_+)^*$. Formally, we define $\mathsf{POS} \subseteq \wp_{<\infty}((\mathbb{N}_+)^*)$ as the set of sets of finite tree positions, such that: $X \in \mathsf{POS}$ iff (a) $X \subseteq (\mathbb{N}_+)^*, |X| < \infty$; (b) X is prefix-closed; and (c) if $n{\cdot}(i+1) \in X$ for $i \in \mathbb{N}_+$, then $n{\cdot}i \in X$. A tree is then a mapping from a set of positions to letters of the alphabet *Trees*$(\mathbb{E}) := \{\mathbf{t} : P \to \mathbb{E} \mid P \in \mathsf{POS}\}$. By $\mathbf{t}|_x(y)$ we denote the subtree of $\mathbf{t}$ at position x: $\mathbf{t}|_x(y) = \mathbf{t}(x{\cdot}y)$. The root's position is the empty string and we denote it by 'ϵ'. The position of any other node in the tree is the concatenation of the position of its parent and the node's index in the ordered list of siblings. Along this work we use x, y, z, w, v as variables for positions, and i, j, k, l, m, n as variables for numbers. For example, $x{\cdot}i$ is a position which is not the root, that has x as parent position, and that has $i - 1$ siblings to the left.

Given a tree $\mathbf{t} \in$ *Trees*$(\mathbb{E})$, $\mathsf{pos}(\mathbf{t})$ denotes the domain of $\mathbf{t}$, which consists of the set of positions of the tree, and $\mathsf{alph}(\mathbf{t}) = \mathbb{E}$ denotes the alphabet of the tree. From now on, we informally refer by 'node' to a position x together with the value $\mathbf{t}(x)$.

Given two trees $\mathbf{t}_1 \in$ *Trees*$(\mathbb{E})$, $\mathbf{t}_2 \in$ *Trees*$(\mathbb{F})$ such that $\mathsf{pos}(\mathbf{t}_1) = \mathsf{pos}(\mathbf{t}_2) = P$, we define $\mathbf{t}_1 \otimes \mathbf{t}_2 : P \to (\mathbb{E}{\times}\mathbb{F})$ as $(\mathbf{t}_1 \otimes \mathbf{t}_2)(x) = (\mathbf{t}_1(x), \mathbf{t}_2(x))$.

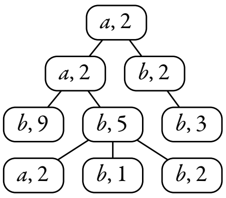


Figure 1 A data tree.

The set of **data trees** over a finite alphabet $\mathbb{A}$ and an infinite domain $\mathbb{D}$ is defined as *Trees*$(\mathbb{A}{\times}\mathbb{D})$. Note that every tree $\mathbf{t} \in$ *Trees*$(\mathbb{A}{\times}\mathbb{D})$ can be decomposed into two trees $\mathbf{a} \in$ *Trees*$(\mathbb{A})$ and $\mathbf{d} \in$ *Trees*$(\mathbb{D})$ such that $\mathbf{t} = \mathbf{a} \otimes \mathbf{d}$. Figure 1 shows an example of a data tree. The notation for the set of data values used in a data tree is *data*$(\mathbf{a} \otimes \mathbf{d}) := \{\mathbf{d}(x) \mid x \in \mathsf{pos}(\mathbf{d})\}$. With an abuse of notation we write *data*(X) to denote all the elements of $\mathbb{D}$ contained in X, for whatever object X may be.

XPath on data trees. Next we define vertical XPath, the fragment of XPath where no horizontal navigation is allowed. We actually consider an extension of XPath allowing the Kleene star on *any* path expression and we denote it by regXPath. Although we define this logic over data trees, our decidability result also holds for the class of XML documents through a standard reduction.

Vertical regXPath is a two-sorted language, with *path* expressions (that we write α, β, γ) and *node* expressions (φ, ψ, η). Path expressions are binary relations resulting from composing the child and parent relations (which are denoted respectively by $\downarrow$ and $\uparrow$), and node expressions. Node expressions are boolean formulas that test a property of a node, like for example, that is has a certain label, or that it has a child labeled a with the same data value as an ancestor labeled b, which is expressed by $\langle\downarrow[a] = \uparrow^*[b]\rangle$. We write regXPath$(\mathfrak{V}, =)$ to denote this logic. A *formula* of regXPath$(\mathfrak{V}, =)$ is either a node expression or a path expression of the logic. Its syntax and semantics are defined in Figure 2. As another example, we can select the nodes that have a descendant labeled b with two children also labeled by

$$\begin{array}{rcll} \alpha, \beta & ::= & o \mid \alpha[\varphi] \mid [\varphi]\alpha \mid \alpha\beta \mid \alpha \cup \beta \mid \alpha^* & o \in \{\varepsilon, \downarrow, \uparrow\}\ , \\ \varphi, \psi & ::= & a \mid \neg\varphi \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \langle\alpha\rangle \mid \langle\alpha = \beta\rangle \mid \langle\alpha \neq \beta\rangle & a \in \mathbb{A}\ . \end{array}$$

$$\begin{array}{rlrl} \llbracket\downarrow\rrbracket^{\mathbf{t}} & = \{(x, x\cdot i) \mid x\cdot i \in \mathsf{pos}(\mathbf{t})\} & \llbracket\uparrow\rrbracket^{\mathbf{t}} & = \{(x\cdot i, x) \mid x\cdot i \in \mathsf{pos}(\mathbf{t})\} \\ \llbracket[\varphi]\rrbracket^{\mathbf{t}} & = \{(x, x) \in\mid x \in \mathsf{pos}(\mathbf{t}), x \in \llbracket\varphi\rrbracket^{\mathbf{t}}\} & \llbracket\alpha^*\rrbracket^{\mathbf{t}} & = \text{the reflexive transitive closure of } \llbracket\alpha\rrbracket^{\mathbf{t}} \\ \llbracket\varepsilon\rrbracket^{\mathbf{t}} & = \{(x, x) \mid x \in \mathsf{pos}(\mathbf{t})\} & \llbracket\alpha\beta\rrbracket^{\mathbf{t}} & = \{(x, z) \mid \text{ there exists } y \text{ such that} \\ \llbracket\alpha \cup \beta\rrbracket^{\mathbf{t}} & = \llbracket\alpha\rrbracket^{\mathbf{t}} \cup \llbracket\beta\rrbracket^{\mathbf{t}} & & \quad (x, y) \in \llbracket\alpha\rrbracket^{\mathbf{t}}, (y, z) \in \llbracket\beta\rrbracket^{\mathbf{t}}\} \\ \llbracket a\rrbracket^{\mathbf{t}} & = \{x \in \mathsf{pos}(\mathbf{t}) \mid \mathbf{a}(x) = a\} & \llbracket\langle\alpha\rangle\rrbracket^{\mathbf{t}} & = \{x \in \mathsf{pos}(\mathbf{t}) \mid \exists y.(x, y) \in \llbracket\alpha\rrbracket^{\mathbf{t}}\} \\ \llbracket\neg\varphi\rrbracket^{\mathbf{t}} & = \mathsf{pos}(\mathbf{t}) \setminus \llbracket\varphi\rrbracket^{\mathbf{t}} & \llbracket\varphi \wedge \psi\rrbracket^{\mathbf{t}} & = \llbracket\varphi\rrbracket^{\mathbf{t}} \cap \llbracket\psi\rrbracket^{\mathbf{t}} \\ \llbracket\langle\alpha{=}\beta\rangle\rrbracket^{\mathbf{t}} & = \{x \in \mathsf{pos}(\mathbf{t}) \mid \exists y,z\ (x, y) \in \llbracket\alpha\rrbracket^{\mathbf{t}}, & \llbracket\langle\alpha{\neq}\beta\rangle\rrbracket^{\mathbf{t}} & = \{x \in \mathsf{pos}(\mathbf{t}) \mid \exists y,z\ (x, y) \in \llbracket\alpha\rrbracket^{\mathbf{t}}, \\ & \quad (x, z) \in \llbracket\beta\rrbracket^{\mathbf{t}}, \mathbf{d}(y) = \mathbf{d}(z)\} & & \quad (x, z) \in \llbracket\beta\rrbracket^{\mathbf{t}}, \mathbf{d}(y) \neq \mathbf{d}(z)\} \end{array}$$

Figure 2 The syntax of XPath($\mathfrak{V}, =$); and its semantics for a data tree $\mathbf{t} = \mathbf{a} \otimes \mathbf{d}$.

b with different data value by a formula $\varphi = \langle\downarrow^*[\, b \wedge \langle\downarrow[b] \neq\downarrow[b]\rangle\,]\rangle$. Given a tree $\mathbf{t}$ as in Figure 1, we have $\llbracket\varphi\rrbracket^{\mathbf{t}} = \{\varepsilon, 1, 12\}$.

The satisfiability problem for regXPath($\mathfrak{V}, =$) is the problem of, given a formula φ, wether there exists a data tree $\mathbf{t}$ such that $\llbracket\varphi\rrbracket^{\mathbf{t}} \neq \emptyset$. Our main result on XPath is the following.

▶ **Theorem 1.** *The satisfiability problem for* regXPath($\mathfrak{V}, =$) *is decidable.*

Well-structured transition systems. The proof of Theorem 1 relies on a translation to an automaton model defined in the next section whose emptiness problem is decidable. This is showed through methods from the theory of *well-structured transition systems*, or WSTS for short [8]. The emptiness test of our model of automata is obtained by interpreting its execution using a transition system compatible with some well-quasi-ordering (wqo). We reproduce here only the result of the theory of WSTS that we will need.

A quasi-order $\leq$ (*i.e.*, a reflexive and transitive relation) over a set S is said to be a **well-quasi-order** (wqo) iff for every infinite sequence $s_1 s_2 \cdots \in S^\omega$ there are two indices $i < j$ such that $s_i \leq s_j$. Given a wqo $(S, \leq)$ and $T \subseteq S$, we define the *downward closure* of T as $\downarrow T := \{s \in S \mid \exists t \in T, s \leq t\}$ and T is **downward closed** if $\downarrow T = T$.

Given a transition system $(S, \to)$, and $T \subseteq S$ we define $Succ(T) := \{s \in S \mid \exists t \in T$ with $t \to s\}$, and $Succ^*$ as its reflexive-transitive closure. We say that $(S, \to)$ is *finitely branching* iff $Succ(\{s\})$ is finite for all $s \in S$. If $Succ(\{s\})$ is also effectively computable for all s, we say that $(S, \to)$ is **effective**.

We adapt some results and definitions of [8, §5] of what is there called *reflexive compatibility* (*i.e.*, compatibility in zero or one steps) to extend them to N-compatibility (*i.e.*, compatibility in at most N steps). Given a binary relation $R \subseteq S \times S$ and $K \subseteq S$, let us write $R^{\leq n}$ for $Id \cup R \cup R^2 \cup \cdots \cup R^n$, where Id is the identity relation and R^i is the i-fold composition of R. Given $N \in \mathbb{N}_+$, a transition system $(S, \to)$ is N**-downward compatible** with respect to a wqo $(S, \leq)$ iff for every $s_1, s_2, s'_1 \in S$ such that $s'_1 \leq s_1$ and $s_1 \to s_2$, there exists $s'_2 \in S$ such that $s'_2 \leq s_2$ and $s'_1\ (\to)^{\leq N}\ s'_2$. In some sense any behavior from the bigger element s_1 can be simulated by the smaller element s'_1. In truth, s'_1 may need several transitions, but not more than N. Hereafter we use the term 'WSTS' to refer to any wqo and transition system N-downward compatible. A simple adaptation of [8, §5] yields the following proposition, what will be used to show decidability for BUDA.

▶ **Proposition 1.** *If $(S, \leq)$ is a* wqo *and $(S, \to)$ a transition system such that (1) it is N-*downwards compatible *for some fixed N, (2) it is effective, and (3) $\leq$ is decidable, $V \subseteq S$ is a recursive downward-closed set and, $T \subseteq S$ is a finite set; then the problem of whether there exist $t \in T$ and $v \in V$ such that $t \to^* v$ is decidable.*

In the above statement one must think of S as the configurations of an automaton, T as its initial configurations, $\to$ as the relation defined by the run, and $\leq$ as a suitable relation between configurations such that the set of accepting configurations V is downward-closed.

3 The automaton model

In this section we introduce the BUDA model. It is essentially a bottom-up tree automaton with one register to store a data value and an alternating control. We show that these automata are at least as expressive as vertical regXPath. In Section 4 we will show that their emptiness problem is decidable. Theorem 1 then follows immediately.

An automaton $\mathcal{A} \in \mathsf{BUDA}$ that runs over data trees of *Trees*$(\mathbb{A} \times \mathbb{D})$ is defined as a tuple $\mathcal{A} = (\mathbb{A}, \mathbb{B}, Q, q_0, \delta_\epsilon, \delta_{up}, \mathcal{S}, h)$ where $\mathbb{A}$ is the finite alphabet of the tree, $\mathbb{B}$ is an internal finite alphabet of the automaton (whose purpose will be clear later), Q is a finite set of states, q_0 is the initial state, $\mathcal{S}$ is a finite semigroup, h is a semigroup homomorphism from $(\mathbb{A} \times \mathbb{B})^+$ to $\mathcal{S}$, δ_ϵ is the ϵ-transition function of $\mathcal{A}$, and δ_{up} is the *up*-transition function of $\mathcal{A}$.

δ_{up} is a partial function from states to formulas. For $q \in Q$, $\delta_{up}(q)$ is either undefined or a formula consisting in a disjunction of conjunctions of states. δ_ϵ is also a partial function from states to disjunctions of conjunctions of 'atoms' of one of the following forms:

$$p \mid \mathsf{guess}(p) \mid \mathsf{univ}(p) \mid \mathsf{store}(p) \mid \mathsf{eq} \mid \overline{\mathsf{eq}} \mid$$
$$\mid \langle\mu\rangle^{=} \mid \langle\mu\rangle^{\neq} \mid \overline{\langle\mu\rangle^{=}} \mid \overline{\langle\mu\rangle^{\neq}} \mid \mathsf{root} \mid \overline{\mathsf{root}} \mid \mathsf{leaf} \mid \overline{\mathsf{leaf}} \mid a \mid \bar{a} \mid b \mid \bar{b}$$

where $\mu \in \mathcal{S}$, $p \in Q$, $a \in \mathbb{A}$, $b \in \mathbb{B}$.

Before we present the precise semantics of our automaton model, here is the intuition. The automaton's control is nondeterministic and alternating, as reflected by the disjunctions and conjunctions in the formulæ specifying the transition functions. Hence, at any node several *threads* of the automaton run in parallel. Each thread consists of a state and a data value stored in the register. At every node of the tree, the automaton guesses a finite internal label of $\mathbb{B}$ and all threads can share access to this finite information. At any node of the tree, the automaton can perform some actions depending on the result of local tests.

We first describe the battery of tests the automata can perform. All these tests are explicitly closed under negation, denoted with the $\overline{\cdot}$ notation, and of course they are also closed under intersection and union using the alternating and nondeterministic control of the automata. The automata can test the label and internal label of the current node and also whether the current node is the root, a leaf or an internal node. The automata can test (in)equality of the current data value with the one stored in the register (eq and $\overline{\mathsf{eq}}$). Finally the automata can test the existence of some downward path, starting from the current node and leading to a node whose data value is (or is not) equal to the one currently stored in the register, such that the path satisfies some regular property on the labels. These properties are specified using the finite semigroup $\mathcal{S}$ and the morphism $h : (\mathbb{A} \times \mathbb{B})^+ \to \mathcal{S}$ over the words made of the label of the tree and the internal label. For example, $\langle\mu\rangle^{=}$ tests for the existence of a path that evaluates to μ via h, which starts at the current node and leads to a node whose data value matches the one currently stored in the register. Similarly, $\langle\mu\rangle^{\neq}$ tests that it leads to a data value different from the one currently in the register. Observe that we could have used finite automata, or regular expressions instead of finite semigroup homomorphisms. We take this approach because it simplifies the notation.

Based on the result of these tests, the automata can perform the following actions. They can change state, store the current data value in the register ($\mathsf{store}(p)$), or store an arbitrary data value nondeterministically chosen ($\mathsf{guess}(p)$). Finally, a transition can demand to start a new thread in state p for every data value of the subtree with the operation $\mathsf{univ}(p)$. The automata can also decide to move up in the tree according to the *up*-transition.

Before we move on to the formal definition, we stress that the automaton model is not closed under complementation because its set of actions are not closed under complementation: guess is a form of existential quantification while univ is a form of universal quantification, but they are not dual. Actually, we will show in the journal version of this paper that adding any of their dual would yield undecidability of the model.

We now turn to the formal definition. A data tree $\mathbf{a}\otimes\mathbf{d} \in \mathit{Trees}(\mathbb{A}\times\mathbb{D})$ is accepted by $\mathcal{A}$ iff there exists an internal labeling $\mathbf{b} \in \mathit{Trees}(\mathbb{B})$ with $\mathsf{pos}(\mathbf{b}) = \mathsf{pos}(\mathbf{a}\otimes\mathbf{d})$ such that there is an accepting run on $\mathbf{a}\otimes\mathbf{b}\otimes\mathbf{d}$. We focus now on the definition of a run.

A **configuration** of a BUDA $\mathcal{A}$ is a set $\mathcal{C}$ of threads, viewed as a finite subset of $Q\times\mathbb{D}$. A configuration $\mathcal{C}$ is said to be **initial** iff it is of the form $\{(q_0, e)\}$ for some $e \in \mathbb{D}$. A configuration $\mathcal{C}$ is **accepting** iff it is empty.

ϵ-transitions. Let $\mathbf{t} = \mathbf{a}\otimes\mathbf{b}\otimes\mathbf{d}$ and $x \in \mathsf{pos}(\mathbf{t})$. Given two configurations $\mathcal{C}$ and $\mathcal{C}'$ of $\mathcal{A}$, we say that there is an ϵ-transition of $\mathcal{A}$ at x between $\mathcal{C}$ and $\mathcal{C}'$, denoted $(x,\mathcal{C}) \rightarrowtail_\epsilon (x,\mathcal{C}')$ (assuming $\mathcal{A}$ and $\mathbf{t}$ are understood from the context) if the following holds: there is a thread with state q and with a data value d (*i.e.*, $(q,d) \in \mathcal{C}$) where $\delta_\epsilon(q) = \bigvee_{i\in I}\gamma_i$. Each γ_i is a conjunction of atoms, and there must be one $i \in I$ with $\gamma_i = \bigwedge_{j\in J}\alpha_j$ and $\mathcal{C}' = (\mathcal{C}\setminus\{(q,d)\})\cup\hat{\mathcal{C}}$ such that the following holds for all $j \in J$:

- If α_j is one of the tests a, b, root, leaf, eq or its negations, it must be true with the obvious semantics as described above.
- If α_j is $\langle\mu\rangle^=$ then there is a downward path in $\mathbf{t}$ starting at x and ending at some descendant position y with $\mathbf{d}(y) = d$, such that the sequence of labels in $\mathbb{A}\times\mathbb{B}$ read while going from x to y along this path (including the endpoints) evaluates to μ via h. The case of $\langle\mu\rangle^{\neq}$ is treated similarly replacing $\mathbf{d}(y) = d$ by $\mathbf{d}(y) \neq d$. The tests $\overline{\langle\mu\rangle^=}$ and $\overline{\langle\mu\rangle^{\neq}}$ correspond to the negation of these tests.
- If α_j is p for some $p \in Q$, then $(p,d) \in \hat{\mathcal{C}}$,
- if α_j is $\mathsf{store}(p)$ then $(p,\mathbf{d}(x)) \in \hat{\mathcal{C}}$,
- if α_j is $\mathsf{guess}(p)$ then $(p,d') \in \hat{\mathcal{C}}$ for some $d' \in \mathbb{D}$,
- if α_j is $\mathsf{univ}(p)$, then for all $d' \in \mathit{data}(\mathbf{t}|_x)$, $(p,d') \in \hat{\mathcal{C}}$,
- nothing else is in $\hat{\mathcal{C}}$.

The **ϵ-closure** of a pair $(x,\mathcal{C})$ is defined as the reflexive transitive closure of $\rightarrowtail_\epsilon$, i.e. the set of configurations reachable from $(x,\mathcal{C})$ by a finite sequence of ϵ-transitions.

***up*-transitions.** We say that a configuration $\mathcal{C}$ is **moving** iff for all $(q,d) \in \mathcal{C}$, $\delta_{up}(q)$ is defined. Given two configurations $\mathcal{C}$ and $\mathcal{C}'$ of $\mathcal{A}$, we say that there is an *up*-transition of $\mathcal{A}$ between $\mathcal{C}$ and $\mathcal{C}'$, denoted $\mathcal{C} \rightarrowtail_{up} \mathcal{C}'$ (assuming $\mathcal{A}$ is understood from the context) if the following conditions hold:

- $\mathcal{C}$ is moving,
- for all $(q,d) \in \mathcal{C}$, if $\delta_{up}(q) = \bigvee_{i\in I}\bigwedge_{j\in J} p_{i,j}$ then there is $i \in I$ such that for all $j \in J$, $(p_{i,j}, d) \in \mathcal{C}'$,
- nothing else is in $\mathcal{C}'$.

▶ Remark. In the definition of the run the automaton behaves synchronously: all threads move up at the same time. This is only for convenience of presentation. Since all the threads are independent, one can also define a run in which each thread moves independently. This alternative definition would be equivalent to the current one.

Runs. We are now ready to define a **run** ρ of $\mathcal{A}$ on $\mathbf{t} = \mathbf{a} \otimes \mathbf{b} \otimes \mathbf{d}$. It is a function associating a configuration to any node x of $\mathbf{t}$ such that

1. for any leaf x of $\mathbf{t}$, $\rho(x) = \{(q_0, \mathbf{d}(x))\}$,
2. for any inner position x of $\mathbf{t}$ whose children are $x{\cdot}1, \dots, x{\cdot}n$, then there are configurations $\mathcal{C}'_1, \cdots, \mathcal{C}'_n$ and $\mathcal{C}''_1, \cdots, \mathcal{C}''_n$ such that for all $i \in [n]$, $(x{\cdot}i, \mathcal{C}''_i)$ is in the ϵ-closure of $(x{\cdot}i, \rho(x{\cdot}i))$, $\mathcal{C}''_i \rightarrowtail_{up} \mathcal{C}'_i$, and $\rho(x) = \bigcup_{i \in [n]} \mathcal{C}'_i$.

The run ρ is **accepting** if moreover at the root (i.e., for the position ϵ), the ϵ-closure of $\rho(\epsilon)$ contains an accepting configuration.

BUDA and vertical regXPath. Given a formula η of regXPath$(\mathfrak{V}, =)$, we say that a BUDA $\mathcal{A}$ is *equivalent* to η if a data tree $\mathbf{t}$ is accepted by $\mathcal{A}$ iff $[\![\eta]\!]^{\mathbf{t}} \neq \emptyset$.

▶ **Proposition 2.** *For every $\eta \in$ regXPath$(\mathfrak{V}, =)$ there exists an equivalent $\mathcal{A} \in$ BUDA computable from η.*

Proof idea. It is easy to simulate any positive test $\langle \alpha = \beta \rangle$ or $\langle \alpha \neq \beta \rangle$ of vertical regXPath by a BUDA using guess, $\langle \mu \rangle^{=}$ and $\langle \mu \rangle^{\neq}$. For example, consider the property $\langle {\downarrow_*}[a] \neq {\uparrow}{\downarrow}[b] \rangle$, which states that there is a descendant labeled a with a different data value than a sibling labeled b. A BUDA automaton can test this property as follows.

1. It guesses a data value d and stores it in the register.
2. It tests that d can be reached by ${\downarrow_*}[a]$ with a test $\langle \mu \rangle^{=}$ for a suitable μ.
3. It moves up to its parent.
4. It tests that a different value than d can be reached in one of its children labeled with b, using the test $\langle \mu \rangle^{\neq}$ for a suitable μ.

The simulation of negative tests ($\neg\langle \alpha = \beta \rangle$ or $\neg\langle \alpha \neq \beta \rangle$) is more tedious as BUDA is not closed under complementation. Nevertheless, the automaton has enough universal quantifications (in the operations univ, $\overline{\langle \mu \rangle^{=}}$ and $\overline{\langle \mu \rangle^{\neq}}$) in order to do the job. Consider for example the formula $\neg\langle {\uparrow^*}[b]{\downarrow}[a] = {\downarrow_*}[c] \rangle$, that states that no data value is shared between a descendant labeled c and an a-child of a b-ancestor. The automaton behaves as follows.

1. It creates one thread in state q for every data value in the subtree, using univ(q).
2. q tests whether the data value of the register is reachable by ${\downarrow_*}[c]$, using a test $\langle \mu \rangle^{=}$. If it is, it changes to state p.
3. p moves up towards the root, and each time it finds a b, it tests that the currently stored data value cannot be reached by ${\downarrow}[a]$. This is done with a test of the kind $\overline{\langle \mu \rangle^{=}}$. ◀

Therefore, in order to conclude the proof of Theorem 1, it remains to show that the emptiness problem of BUDA is decidable. This is the goal of Section 4.

Automata normal form. We now present a normal form of BUDA, removing all the redundancy in its definition. This normal form simplifies the technical details in the proof of decidability presented in the next section.

(NF1) The semigroup $\mathcal{S}$ and morphism h have the following property. For all $w \in (\mathbb{A} \times \mathbb{B})^+$ and $c \in \mathbb{A} \times \mathbb{B}$, $h(w) = h(c)$ iff $w = c$.

(NF2) In the definition of δ_{up} of $\mathcal{A}$, there is exactly one disjunct that contains exactly one conjunct. That is, for all $q \in Q$, $\delta_{\text{up}}(q)$ is undefined or $\delta_{\text{up}}(q) = p$ for some $p \in Q$.

(NF3) For all $q \in Q$, $\delta_{\epsilon}(q)$ is defined either as an atom, as $p \wedge p'$ or as $p \vee p'$ for some $p, p' \in Q$.

(NF4) For all $q \in Q$, $\delta_{\epsilon}(q)$ does not contain tests for labels (a, $\bar{a}$, b, $\bar{b}$), eq, $\overline{\text{eq}}$, store, leaf or $\overline{\text{leaf}}$.

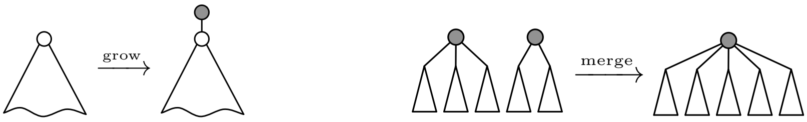


Figure 3 The *grow* and *merge* operations.

An automaton $\mathcal{A} \in$ BUDA is said to be in **normal form** if it satisfies (NF1), (NF2), (NF3) and (NF4). Notice that once (NF1) holds, then any test concerning a label (a, $\bar{a}$, b, or $\bar{b}$) can be simulated using tests of the form $\langle\mu\rangle$ for some appropriate μ. Using similar ideas, it is not hard to check that:

▶ **Proposition 3.** *For any $\mathcal{A} \in$ BUDA, there is an equivalent $\mathcal{A}' \in$ BUDA in normal form that can be effectively obtained.*

4 The emptiness problem for BUDA

The goal of this section is to show:

▶ **Theorem 2.** *The emptiness problem for* BUDA *is decidable.*

In order to achieve this, we associate with each BUDA a WSTS that simulates its runs. The transition system works on sets of *abstract configurations*. Given an automaton, an abstract configuration consists of all the information of the run that is necessary to maintain at the root of a given subtree in order to continue the simulation of the automaton from there. The aforesaid transition system works with *sets* of such abstract configurations in order to capture the bottom-up behavior of the automaton on unranked trees. The transition relation of the WSTS essentially corresponds to the transitions of the automaton except for the *up*-transition. An *up*-transition of the automaton is simulated by a succession of two types of transitions of the WSTS, called *grow* and *merge*. The object of doing this is to avoid having transitions that take an unbounded number of arguments (as the *up* relation in the run of the automaton does). The *grow* transition adds a node on top of the current root, and the *merge* transition identifies the roots of two abstract configurations. Intuitively, these transitions correspond to the operations on trees of Figure 3. This is necessary because we do not know in advance the arity of the tree and therefore the transition system has to build one subtree at a time. We then exhibit a wqo on abstract configurations and show that the transition system is N-downward compatible with respect to this wqo for some N that depends on the automaton. Decidability will then follow from Proposition 1.

In the sequel we implicitly assume that all our BUDA are in normal form.

4.1 Abstract configurations

Given a BUDA $\mathcal{A} = (\mathbb{A}, \mathbb{B}, Q, q_0, \delta_\epsilon, \delta_{up}, \mathcal{S}, h)$, we start the definition of its associated WSTS by defining its universe: finite sets of abstract configurations of $\mathcal{A}$.

An **abstract configuration** of $\mathcal{A}$ is a tuple (Δ, Γ, r, m) where r and m are either true or false, Δ is a finite subset of $Q \times \mathbb{D}$ and Γ is a finite subset of $\mathcal{S} \times \mathbb{D}$ such that

$$\Gamma \text{ contains exactly one pair of the form } (h(c), d) \text{ with } c \in \mathbb{A} \times \mathbb{B}. \qquad (\star)$$

This unique element of $\mathbb{A} \times \mathbb{B}$ is denoted as the **label** of the abstract configuration and the unique associated data value is denoted as the **data value** of the abstract configuration.

Intuitively r says whether the current node should be treated as the root or not, m says whether we are in a phase of *merging configurations* or not (a notion that will become clear when we introduce our transition system later on), Δ is the set of ongoing threads (corresponding to the configuration of the automaton) and a pair $(\mu, d) \in \Gamma$ simulates the existence of a downward path evaluating to μ and whose last element carries the datum d.

In the sequel we will use the following notation: $\Delta(d) = \{q \mid (q, d) \in \Delta\}$, $\Gamma(d) = \{\mu \mid (\mu, d) \in \Gamma\}$, $\Delta(q) = \{d \mid (q, d) \in \Delta\}$, $\Gamma(\mu) = \{d \mid (\mu, d) \in \Gamma\}$. We also use the notation $\Delta \otimes \Gamma : \mathbb{D} \to \wp(Q) \times \wp(\mathcal{S})$ with $(\Delta \otimes \Gamma)(d) = (\Delta(d), \Gamma(d))$. Given a data value d and an abstract configuration θ, $(\Delta \otimes \Gamma)(d)$ is also denoted as the **type of** d **in** θ. We use the letter θ to denote an abstract configuration and we write AC to denote the set of all abstract configurations. Similarly, we use Θ to denote a finite set of abstract configurations and $\wp_{<\infty}(\text{AC})$ for the set of finite sets of abstract configurations.

An abstract configuration $\theta = (\Delta, \Gamma, r, m)$ is said to be **initial** if it corresponds to a leaf node, i.e., is such that $\Delta = \{(q_0, d)\}$ and $\Gamma = \{(h(a), d)\}$ for some $d \in \mathbb{D}$ and $a \in \mathbb{A} \times \mathbb{B}$. It is said to be **accepting** if Δ is empty and r is *true*.

Two configurations θ_1 and θ_2 are said to be **equivalent** if there is a bijection $f : \mathbb{D} \to \mathbb{D}$ such that $f(\theta_1) = \theta_2$ (with some abuse of notation). In this case we note $\theta_1 \sim \theta_2$.

Finally, we write Θ_I to denote the set of all initial abstract configurations modulo $\sim$ (i.e., a set containing at most one element for each $\sim$ equivalence class). Note that Θ_I is *finite* and effective. A set of abstract configurations is said to be **accepting** iff it contains an accepting abstract configuration.

4.2 Well-quasi-orders

We now equip $\wp_{<\infty}(\text{AC})$ with a well-quasi-order $(\wp_{<\infty}(\text{AC}), \leq_{min})$. The order $\leq_{min}$ builds upon a wqo $(\text{AC}, \precsim)$ over abstract configurations. Let us define these orderings precisely.

The **profile of an abstract configuration** $\theta = (\Delta, \Gamma, r, m)$, denoted by *profile*$(\theta)$, is *profile*$(\theta) = (\ A_0,\ A_1,\ r,\ m\ )$ with $A_i = \{(S, \chi) \in \wp(Q) \times \wp(\mathcal{S}) : |(\Delta \otimes \Gamma)^{-1}(S, \chi)| = i\}$.

We first define the quasi-order $\preceq$ over abstract configurations, and then we define the order $(\text{AC}, \precsim)$ as $(\text{AC}, \preceq)$ modulo $\sim$. Given two abstract configurations $\theta_1 = (\Delta_1, \Gamma_1, r_1, m_1)$ and $\theta_2 = (\Delta_2, \Gamma_2, r_2, m_2)$, we denote by $\theta_1 \preceq \theta_2$ the fact that

- *profile*$(\theta_1) =$ *profile*(θ_2), and
- $(\Delta_1 \otimes \Gamma_1) \subseteq (\Delta_2 \otimes \Gamma_2)$.

▶ Remark. Notice that due to condition $(\star)$, $\theta_1 \preceq \theta_2$ implies that θ_1 and θ_2 have the same label and same data value.

We now define $\precsim$ as: $\theta_1 \precsim \theta_2$ iff $\theta_1' \preceq \theta_2$ for some $\theta_1' \sim \theta_1$.

We are now ready to define our wqo over $\wp_{<\infty}(\text{AC})$. Given Θ_1 and Θ_2 in $\wp_{<\infty}(\text{AC})$ we define $\leq_{min}$ as: $\Theta_1 \leq_{min} \Theta_2$ iff for all $\theta_2 \in \Theta_2$ there is $\theta_1 \in \Theta_1$ such that $\theta_1 \precsim \theta_2$. That is, every element from Θ_2 is *minorized* by an element of Θ_1.

The following is a key observation:

▶ **Lemma 3.** $(\wp_{<\infty}(\text{AC}), \leq_{min})$ *is a* wqo.

Finally, the following obvious lemma will be necessary to apply Proposition 1.

▶ **Lemma 4.** $\{\Theta \in \wp_{<\infty}(\text{AC}) \mid \Theta$ *is accepting*$\}$ *is downward closed for* $(\wp_{<\infty}(\text{AC}), \leq_{min})$.

4.3 Transition system

We now equip $\wp_{<\infty}(\text{AC})$ with a transition relation $\Rightarrow$. This transition relation is built upon a transition relation $\to$ over AC.

Let us first define $\rightarrow$ over AC. It is specified so that it reflects the transitions of $\mathcal{A}$. For each possible test of the automaton there is a transition that simulates this test by using the information contained in Γ, and removes the corresponding in Δ. And for every operation store, guess, univ there is a transition that modifies Δ. We call the ϵ-transitions of $\rightarrow$, that we note $\rightarrow_\epsilon$. On the other hand, any *up* transition of $\mathcal{A}$ is decomposed into transitions $\xrightarrow{\text{merge}}$ and $\xrightarrow{\text{grow}}$ that modify not only Δ but also Γ accordingly.

We start with ϵ-transitions. Given two abstract configurations $\theta_1 = (\Delta_1, \Gamma_1, r_1, m_1)$ and $\theta_2 = (\Delta_2, \Gamma_2, r_2, m_2)$, we say that $\theta_1 \rightarrow_\epsilon \theta_2$ if $m_1 = m_2 = \mathit{false}$ (the merge information is used for simulating an *up*-transition as will be explained later), $r_2 = r_1$ (whether the current node is the root or not should not change), θ_1 and θ_2 have the same label and data value, $\Gamma_2 = \Gamma_1$ (the tree is not affected by an ϵ-transition) and, furthermore, one of the following conditions holds:

1. $\theta_1 \xrightarrow{\mathsf{univ}} \theta_2$. This transition can happen if there is $(q, d) \in \Delta_1$ with $\delta_\epsilon(q) = \mathsf{univ}(p)$ for some $p, q \in Q$. In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, d)\}) \cup \{(p, e) : \exists\mu\ .\ (\mu, e) \in \Gamma_1\}$.
2. $\theta_1 \xrightarrow{\mathsf{guess}} \theta_2$. This transition can happen if there is $(q, d) \in \Delta_1$ with $\delta_\epsilon(q) = \mathsf{guess}(p)$ for some $p, q \in Q$. In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, d)\}) \cup \{(p, d')\}$ for some $d' \in \mathbb{D}$.
3. $\theta_1 \xrightarrow{\langle\mu\rangle^=} \theta_2$ (resp. $\theta_1 \xrightarrow{\langle\mu\rangle^{\neq}} \theta_2$). This transition can happen if there is $(q, d) \in \Delta_1$ with $\delta_\epsilon(q) = \langle\mu\rangle^=$ (resp. $\delta_\epsilon(q) = \langle\mu\rangle^{\neq}$) for some $q \in Q$, $\mu \in \mathcal{S}$, and $\mu \in \Gamma_1(d)$ (resp. there exists $e \in \mathbb{D}$, $e \neq d$ such that $\mu \in \Gamma_1(e)$). In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, d)\})$. The negation of these tests $\xrightarrow{\overline{\langle\mu\rangle^=}}$ and $\xrightarrow{\overline{\langle\mu\rangle^{\neq}}}$ are defined in a similar way.
4. $\theta_1 \xrightarrow{\mathsf{root}} \theta_2$ (resp. $\theta_1 \xrightarrow{\overline{\mathsf{root}}} \theta_2$). This transition can happen if there is $(q, d) \in \Delta$ with $\delta_\epsilon(q) = \mathsf{root}$ and $r_1 = \mathit{true}$ (resp. $\delta_\epsilon(q) = \overline{\mathsf{root}}$ and $r_1 = \mathit{false}$). In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, d)\})$.
5. $\theta_1 \xrightarrow{\wedge} \theta_2$. This transition can happen if there is $(q, d) \in \Delta_1$ with $\delta_\epsilon(q) = p \wedge p'$ for some $p, p', q \in Q$. In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, d)\}) \cup \{(p, d), (p', d)\}$.
6. $\theta_1 \xrightarrow{\vee} \theta_2$. This transition can happen if there is $(q, d) \in \Delta_1$ with $\delta_\epsilon(q) = p \vee p'$ for some $p, p', q \in Q$. In this case θ_2 is such that $\Delta_2 = (\Delta_1 \setminus \{(q, d)\}) \cup A$, for $A = \{(p, d)\}$ or $A = \{(p', d)\}$.

Note that by (NF3) and (NF4) for every possible definition of $\delta_\epsilon(q)$ there is one transition that simulates it. To simulate δ_{up}, it turns out that we will need one extra ϵ-transition that makes our trees fatter. This transition assumes the same constraints as for $\rightarrow_\epsilon$ except that we no longer have $\Gamma_2 = \Gamma_1$. The idea is that this transition corresponds to duplicating all the immediate subtrees of the root. For example, if the root has $\mathbf{t}_1$ and $\mathbf{t}_2$ as subtrees, consider the operation of now having $\mathbf{t}_1\ \mathbf{t}_2\ \mathbf{t}_1'\ \mathbf{t}_2'$ as subtrees, where $\mathbf{t}_1'$ and $\mathbf{t}_2'$ are identical to $\mathbf{t}_1$ and $\mathbf{t}_2$ except for one data value with profile (S, χ) that in $\mathbf{t}_1'$ and $\mathbf{t}_2'$ is replaced by a fresh data value. Here is the definition that follows this idea in terms of our transition system. We say that $\theta_1 \xrightarrow{\mathrm{inc}(S,\,\chi)} \theta_2$ for some pair $(S, \chi) \in \wp(Q) \times \wp(\mathcal{S})$ if $|(\Delta_1 \otimes \Gamma_1)^{-1}(S, \chi)| \geq 1$ and, either $\chi = \emptyset$ or $|(\Gamma_1)^{-1}(\chi)| \geq 2$. Then θ_2 is such that $\mathit{data}(\theta_2) = \mathit{data}(\theta_1) \cup \{e\}$ for some $e \notin \mathit{data}(\theta_1)$, $(\Delta_2 \otimes \Gamma_2)(e) = (S, \chi)$, and for all $d \neq e$, $(\Delta_2 \otimes \Gamma_2)(d) = (\Delta_1 \otimes \Gamma_1)(d)$. Observe that $\xrightarrow{\mathrm{inc}(S,\,\chi)}$ does not change the truth value of any test. Indeed, any test ($\langle\mu\rangle^=$, $\overline{\langle\mu\rangle^=}$, eq, etc.) that is true in θ, continues to be true after a $\xrightarrow{\mathrm{inc}(S,\,\chi)}$ transition, and vice-versa.

We define the transitions of the WSTS that correspond to *up*-transitions in the automaton. We split them into two phases: adding a new root symbol and merging the roots.

1. $\theta_1 \xrightarrow{\text{grow}} \theta_2$. Given two abstract configurations θ_1 and θ_2 as above, we say $\theta_1 \xrightarrow{\text{grow}} \theta_2$ if $r_1 = m_1 = \mathit{false}$, and for all $(q, d) \in \Delta_1$, $\delta_{up}(q)$ is defined and θ_2 is such that $\Delta_1 \rightarrowtail_{up} \Delta_2$,

and $\Gamma_2 = \{(\mu', e) : (\mu, e) \in \Gamma_1, \mu' = h(c)\cdot\mu\} \cup \{(h(c), d)\}$, for some $c \in \mathbb{A} \times \mathbb{B}$ and $d \in \mathbb{D}$. Notice that c and d are then the label and data value of θ_2. As a consequence of the normal form (NF1) of the semigroup, this operation preserves property ($\star$).

2. $\theta_1, \theta_2 \xrightarrow{\text{merge}} \theta_0$. Given 3 abstract configurations $\theta_1 = (\Delta_1, \Gamma_1, r_1, m_1)$, $\theta_2 = (\Delta_2, \Gamma_2, r_2, m_2)$, $\theta_0 = (\Delta_0, \Gamma_0, r_0, m_0)$ we define $\theta_1, \theta_2 \xrightarrow{\text{merge}} \theta_0$ if they all have the same label and data value, $m_1 = m_2 = \mathit{true}$, $r_1 = r_2 = r_0$, $\Delta_0 = \Delta_1 \cup \Delta_2$, and $\Gamma_0 = \Gamma_1 \cup \Gamma_2$. Notice that this operation preserves property ($\star$).

▶ Remark. $\xrightarrow{\text{inc}(S,\,\chi)}$ can be seen as a kind of $\xrightarrow{\text{merge}}$ which preserves the truth of tests.

We are now ready to define the transition relation over $\wp_{<\infty}(\mathrm{AC})$.

▶ **Definition 5.** We define that $\Theta_1 \Rightarrow \Theta_0$ if one the following conditions holds:

1. There is $\theta_1 \in \Theta_1$ and $\theta'_1 \sim \theta_1$ such that $\theta'_1 \rightarrow_\epsilon \theta_0$ or $\theta'_1 \xrightarrow{\text{inc}(S,\,\chi)} \theta_0$ or $\theta'_1 \xrightarrow{\text{grow}} \theta_0$, for some θ_0, χ, and $\Theta_0 = \Theta_1 \cup \{\theta_0\}$.
2. There are $\theta_1, \theta_2 \in \Theta_1$ and $\theta'_1 \sim \theta_1$, $\theta'_2 \sim \theta_2$ such that $\theta'_1, \theta'_2 \xrightarrow{\text{merge}} \theta_0$ for some θ_0, and $\Theta_0 = \Theta_1 \cup \{\theta_0\}$.

In the definition of the transition system, the m flag is simply used to constrain the transition system to have all its $\xrightarrow{\text{merge}}$ operations right after $\xrightarrow{\text{grow}}$ and before any $\rightarrow_\epsilon$. Thus, if we take a derivation and examine the kind of $\rightarrow$ transitions that originated each $\Rightarrow$ transition, we obtain a word described by the following regular expression

$$\Big((\rightarrow_\epsilon \mid \xrightarrow{\text{inc}(S,\,\chi)})^* \xrightarrow{\text{grow}} (\xrightarrow{\text{merge}})^*\Big)^* (\rightarrow_\epsilon \mid \xrightarrow{\text{inc}(S,\,\chi)})^* \ . \qquad (\dagger)$$

4.4 Compatibility

We now show that all the previous definitions were chosen appropriately and that the transition system defined in Section 4.3 is compatible with the wqo defined in Section 4.2. The proof of this result is very technical and consists in a case analysis over each possible kind of transition. In this proof, the operation $\xrightarrow{\text{inc}(S,\,\chi)}$ becomes crucial to show that the downwards compatibility can always be done in a bounded amount of N steps. The detailed proof will appear in the journal version of this paper.

▶ **Proposition 4.** *The transition system* $(\wp_{<\infty}(\mathrm{AC}), \Rightarrow)$ *is* $\mathbf{N}$*-downward compatible with respect to* $(\wp_{<\infty}(\mathrm{AC}), \leq_{min})$*, for* $\mathbf{N} := 2 \cdot (|S| \cdot |Q|)^2 + 1$.

Let $\equiv$ be the equivalence relation over $\wp_{<\infty}(\mathrm{AC})$ such that $\Theta \equiv \Theta'$ iff $\Theta \leq_{min} \Theta'$ and $\Theta' \leq_{min} \Theta$. Given a BUDA $\mathcal{A}$, the WSTS $(\wp_{<\infty}(\mathrm{AC})/{\equiv}, \Rightarrow, \leq_{min})$ as built in the previous section is called *the* WSTS *associated with* $\mathcal{A}$. From Proposition 1 and 4 we obtain:

▶ **Corollary 6.** *Given a* BUDA $\mathcal{A}$*, it is decidable whether the* WSTS *associated with* $\mathcal{A}$ *can reach an accepting abstract configuration from its initial abstract configuration.*

As shown next, this implies the decidability for the emptiness problem for BUDA.

4.5 From BUDA to its abstract configurations

As expected, the WSTS associated with a BUDA $\mathcal{A}$ reflects its behavior. That is, reachability of one corresponds exactly to accessibility of the other. One direction is easy as the transition system can easily simulate $\mathcal{A}$. The other direction requires more care. As evidenced in ($\dagger$), the WSTS may perform a $\xrightarrow{\text{inc}(S,\,\chi)}$ transition anytime. However, BUDA can only make the tree grow in width when moving up in the tree. This issue is solved by showing that all other transitions commute with $\xrightarrow{\text{inc}(S,\,\chi)}$. Finally we obtain the following.

▶ **Proposition 5.** *Let $\mathcal{A}$ be a* BUDA*. Let $\mathcal{W}$ be the* WSTS *associated with $\mathcal{A}$. Then $\mathcal{A}$ has an accepting run iff $\mathcal{W}$ can reach an accepting set of abstract configurations from the initial set of abstract configurations.*

Hence, combining Proposition 5 and Corollary 6 we prove Theorem 2.

5 Concluding remarks

We have exhibited a decidable class of automata over data trees. This automaton model is powerful enough to code node expressions of regXPath($\mathfrak{V}$, =). Therefore, since these expressions are closed under negation, we have shown decidability of the satisfiability, inclusion and equivalence problems for node expressions of regXPath($\mathfrak{V}$, =).

Notice that if our result implies also the decidability of the emptiness problem for **path** expressions of regXPath($\mathfrak{V}$, =), those being not closed under complementation it is not clear that their inclusion or equivalence problem remains decidable.

Our decision algorithm relies heavily on the fact that we work with **unranked** data trees. As already shown in [7] without this assumption XPath($\mathfrak{V}$, =) would be undecidable. In particular if we further impose the presence of a DTD, XPath($\mathfrak{V}$, =) becomes undecidable.

Finally we remark that our decision algorithm is not primitive recursive. But as shown in [7] there cannot be a primitive recursive decision algorithm for XPath($\mathfrak{V}$, =).

References

1 Michael Benedikt, Wenfei Fan, and Floris Geerts. XPath satisfiability in the presence of DTDs. *Journal of the ACM*, 55(2):1–79, 2008.

2 Michael Benedikt and Christoph Koch. XPath leashed. *ACM Computing Surveys*, 41(1), 2008.

3 Mikołaj Bojańczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *Journal of the ACM*, 56(3):1–48, 2009.

4 Diego Figueira. Satisfiability of downward XPath with data equality tests. In *ACM Symposium on Principles of Database Systems (PODS'09)*, 2009.

5 Diego Figueira. Forward-XPath and extended register automata on data-trees. In *International Conference on Database Theory (ICDT'10)*, 2010.

6 Diego Figueira. *Reasoning on words and trees with data.* PhD thesis, ÉNS de Cachan, 2010. Available at `http://www.lsv.ens-cachan.fr/~figueira/phd/`.

7 Diego Figueira and Luc Segoufin. Future-looking logics on data words and trees. In *Intl. Symp. on Mathematical Foundations of Computer Science (MFCS'09)*, 2009.

8 Alain Finkel and Philippe Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.

9 Floris Geerts and Wenfei Fan. Satisfiability of XPath queries with sibling axes. In *Intl. Symp. on Database Programming Languages (DBPL'05)*, 2005.

10 Georg Gottlob, Christoph Koch, and Reinhard Pichler. Efficient algorithms for processing XPath queries. *ACM Transactions on Database Systems*, 30(2):444–491, 2005.

11 Petr Jančar. A note on well quasi-orderings for powersets. *Information Processing Letters*, 72(5-6):155–160, 1999.

12 Marcin Jurdziński and Ranko Lazić. Alternation-free modal mu-calculus for data trees. In *Logic in Computer Science (LICS'07)*, 2007.

13 Alberto Marcone. Foundations of bqo theory. *Transactions of the American Mathematical Society*, 345:641–660, 1994.

Data Monoids*

Mikolai Bojańczyk[1]

1 **University of Warsaw**

Abstract

We develop an algebraic theory for languages of data words. We prove that, under certain conditions, a language of data words is definable in first-order logic if and only if its syntactic monoid is aperiodic.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Monoid, Data Words, Nominal Set, First-Order Logic

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.105

1 Introduction

This paper is an attempt to combine two fields.

The first field is the algebraic theory of regular languages. In this theory, a regular language is represented by its syntactic monoid, which is a finite monoid. It turns out that many important properties of the language are reflected in the structure of its syntactic monoid. One particularly beautiful result is the Schützenberger-McNaughton-Papert theorem, which describes the expressive power of first-order logic.

> Let $L \subseteq A^*$ be a regular language. Then L is definable in first-order logic if and only if its syntactic monoid M_L is aperiodic.

For instance, the language "words where there exists a position with label a" is defined by the first-order logic formula (this example does not even use the order on positions $<$, which is also allowed in general)

$$\exists x.\ a(x).$$

The syntactic monoid of this language is isomorphic to $\{0, 1\}$ with multiplication, where 0 corresponds to the words that satisfy the formula, and 1 to the words that do not. Clearly, this monoid does not contain any non-trivial group. There are many results similar to theorem above, each one providing a connection between seemingly unrelated concepts of logic and algebra, see e.g. the book [8].

The second field is the study of languages over infinite alphabets, commonly called languages of data words. Regular languages are usually considered for finite alphabets. Many current motivations, including XML and verification, require the use of infinite alphabets. In this paper, we use the following definition of data words. We fix for the rest of the paper a single infinite alphabet D, and study words and languages over D. A typical language, which we use as our running example, is "some two consecutive positions have the same letter", i.e.

$$L_{dd} = \bigcup_{d \in D} D^* dd D^* = \{d_1 \cdots d_n \in D^* : d_i = d_{i+1} \text{ for some } i \in \{1, \ldots, n-1\}\}.$$

* Author supported by ERC Starting Grant "Sosna".

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 105–116
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

A number of automata models have been developed for such languages. The general theme is that there is a tradeoff between the following three properties: expressivity, good closure properties, and decidable (or even efficiently decidable) emptiness. The existing models strike different balances in this tradeoff, and it is not clear which balance, if any, should deserve the name of "regular language". Also, logics have been developed to express properties of data words. For more information on automata and logics for data words, see the survey [7].

The motivation of this paper is to combine the two fields, and prove a theorem that is analogous to the Schützenberger-McNaughton-Papert characterization theorem, but talks about languages of data words. If we want an analogue, we need to choose the notion of regular language for data words, the definition of first-order logic for data words, and then find the property corresponding to aperiodicity.

For first-order logic over data words we use a notion that has been established in the literature. Formulas are evaluated in data words. The quantifiers quantify over positions of the data word, we allow two binary predicates: $x < y$ for order on positions, and $x \sim y$ for having the same data value. An example formula of a formula of first-order logic is the formula

$$\exists x \exists y \qquad x < y \quad \wedge \quad x \sim y \quad \wedge \quad \neg(\exists z\; x < z \wedge z < y),$$

which defines the language L_{dd} in the running example.

As for the notion of regular language, we use the monoid approach. For languages of data words, we use the syntactic monoid, defined in the same way as it is for words over finite alphabets. That is, elements of the syntactic monoid are equivalence classes of the two-sided Myhill-Nerode congruence. When the alphabet is infinite, the syntactic monoid is infinite for every language of data words, except those that depend only on the length of the word. For instance, in the case of the running example language L_{dd}, two words $w, w' \in D^*$ are equivalent if and only if: either both belong to L_{dd}, or both have the same first and last letters. Since there are infinitely many letters, there are infinitely many equivalence classes.

Instead of studying finite monoids, we study something called *orbit finite* monoids. Intuitively speaking, two elements of a syntactic monoid are considered to be in the same orbit if there is a renaming of data values that maps one element to the other[1]. For instance, in the running example L_{dd}, the elements of the syntactic monoid that correspond to the words $1 \cdot 7$ and $2 \cdot 3 \cdot 4 \cdot 8$ are not equal, but they are in the same orbit, because the renaming $i \mapsto i + 1$ maps $1 \cdot 7$ to $2 \cdot 8$, which corresponds to the same element of the syntactic monoid as $2 \cdot 3 \cdot 4 \cdot 8$. It is not difficult to see that the syntactic monoid of L_{dd} has four orbits: one for the empty word, one for words inside L_{dd}, one for words outside L_{dd} where the first and last letters are equal, and one for words outside L_{dd} where the first and last letters are different.

The contribution of this paper is a study of orbit finite data monoids. We develop the algebraic theory of orbit finite data monoids, and show that it resembles the theory of finite monoids. The main result of the paper is Theorem 11, which shows that the Schützenberger-McNaughton-Papert characterization also holds for languages of data words with orbit finite syntactic data monoids:

> Let $L \subseteq D^*$ be a language whose syntactic data monoid M_L is orbit finite. Then L is definable in first-order logic if and only if M_L is aperiodic.

[1] This is related to Proposition 3 in [6]

Related work. The idea to present effective characterizations of logics on data words was proposed by Benedikt, Ley and Puppis. In [1], they show that definability in first-order logic is undecidable if the input is a nondeterministic register automaton. Also, they provide some decidable characterizations, including a characterization of first-order logic with local data comparisons within the class of languages recognized by deterministic register automata. (This result is incomparable the one in this paper.)

The key property of data monoids, namely that elements of the monoid are acted on by permutations of data values, appears in the nominal sets of Gabbay and Pitts [5]. The connections with the literature on nominal sets are still relatively unexplored.

There are two papers on algebra for languages over infinite alphabets. One approach is described by Bouyer, Petit and Thérien [3]. Their monoids are different from ours in the following ways: the definition of a monoid explicitly talks about registers, there is no syntactic monoid, monoids have undecidable emptiness (not mention questions such as aperiodicity or first-order definability). Our setting is closer to the approach of Francez and Kaminski from [4], but the latter talks about automata and not monoids, and does not study the connection with logics.

Acknowledgments. I would like to thank Clemens Ley for introducing me to the subject, Sławomir Lasota for suggesting the key idea of orbits, and Bartek Klin for pointing out the connection to nominal sets. I would also like to thank Michael Kaminski and the anonymous referees for their many helpful comments.

2 Myhill-Nerode equivalence

We start out with an investigation of the Myhill-Nerode syntactic congruence, in the case of data words.

We fix for the rest of this paper an infinite set of data values D. One can think of D as being the natural numbers, but all the structure of natural numbers (such as order) is not going to play a role. Data words are words in D^*. Inside D, we will distinguish a finite subset $C \subseteq D$ of constants. Although D is fixed, the constants will vary. The constants are introduced for two purposes. First, interpreting a finite alphabet as a set of constants, one recovers the traditional theory of regular languages in this setting. Second, constants are useful in the proofs. All definitions will use the set of constant as a parameter.

We use the term *data renaming (with constants C)* for a bijection on the alphabet that is the identity function on C.[2] To simplify proofs, we require a data renaming to be the identity on all but finitely many data values[3]. We write Γ_C for the set of all data renamings with constants C. This set has a group structure. We write π, τ, σ for data renamings. When we apply a data renaming π to a data value d, we skip the brackets and write πd instead of $\pi(d)$.

A data renaming π is naturally extended to a homomorphism $[\pi] : D^* \to D^*$ of data words. For a fixed set of constants C, we study only languages that are invariant under data renaming, i.e. languages such that $[\pi]L = L$ for any data renaming $\pi \in \Gamma_C$.[4]

Suppose that $L \subseteq D^*$ is a language that is invariant under data renamings. We define two equivalence relations on D^*.

[2] This is called C-preservation in Definition 3 of [4].

[3] The same results hold when all bijections are allowed, including the Memory Theorem.

[4] These languages are called co-C-invariant languages in Definition 8 of [4].

The first equivalence relation is the two-sided Myhill-Nerode equivalence relation:

$$w \equiv_L v \qquad \text{if for all } u_1, u_2 \in D^*,\ u_1 w u_2 \in L \Leftrightarrow u_1 v u_2 \in L.$$

We also use the name *syntactic congruence* for this relation. If L uses only constants, i.e. $L \subseteq C^*$, then this equivalence relation has finitely many equivalence classes. However, for any language that uses non-constants in a nontrivial way, this equivalence relation will have infinitely many equivalence classes. The problem is that the relation really cares about specific data values.

This problem is fixed by the second equivalence relation:

$$w \simeq_L v \qquad \text{if for some data renaming } \tau \in \Gamma_C,\ [\tau]w \equiv_L v.$$

As discussed in the introduction, in the running example L_{dd} this equivalence relation has four equivalence classes. On the other hand, some reasonable languages have infinitely many equivalence classes in $\simeq_L$. An example of such an L is "the first letter also appears one some other position". This language is recognized by a left-to-right register automaton with three states. Nevertheless, an equivalence class of a word w under $\equiv_L$ is determined by the first letter of w and the set of distinct letters in w. Consequently, any two words with a different number of distinct letters are not equivalent under $\simeq_L$. (This problem disappears when one uses the one-sided Myhill-Nerode equivalence.)

Despite the limitations, we will be studying languages that have finitely many equivalence classes for $\simeq_L$, because they yield a monoid structure.

3 Data monoids

Fix a finite set of constants $C \subseteq D$. We want to define something like a monoid for data languages. We want there to be a "free object", which should describe D^*. We want each language to have its syntactic object, which should be finite in many interesting cases, and be somehow optimal for all other objects capturing the language. These goals are achieved by the notion of a data monoid, as defined below.

Definition of a data monoid. A *data monoid* (with constants C) is a monoid M, which is acted upon by data renamings (with constants C). That is, for every data renaming $\tau \in \Gamma_C$, there is an associated monoid automorphism $[\tau] : M \to M$. The mapping $\tau \mapsto [\tau]$ should be a group action, namely $[\tau \circ \sigma] = [\tau] \circ [\sigma]$, and the identity renaming should be associated to the identity automorphism. There are two key examples:

- The free data monoid. The monoid is D^*, while the action is defined by

$$[\tau](d_1 \cdots f_n) = \tau(d_1) \cdots \tau(d_n) \qquad \text{for } d_1, \ldots, d_n \in D.$$

- The syntactic data monoid of a language $L \subseteq D^*$, which we denote M_L. The monoid is the quotient of D^* under the Myhill-Nerode equivalence $\equiv_L$. This is a well defined monoid, since $\equiv_L$ is a monoid congruence (unlike $\simeq_L$). The action is defined by

$$[\tau]([w]_{/\equiv_L}) = [[\tau](w)]_{/\equiv_L}.$$

  In Lemma 1, we show that this action is well defined and does not depend on w.

In the sequel, we will drop the brackets $[\tau]$ and simply write τ, when the context indicates if τ is treated as function on data values, or on elements of the monoid. Note that for two renamings τ, σ and an element m of the monoid, the notation $\tau\sigma m$ is unambiguous, since we are dealing with an action.

Finite support axiom. We say that a set $X \subseteq D$ of data values *supports* an element m of a data monoid, if $\tau m = m$ holds for every data renaming in $\tau \in \Gamma_X$. We require an additional property from a data monoid, called the *finite support axiom*: for any element m of the data monoid, there is a finite support.

Both examples above satisfy the finite support axiom. The axiom is violated by some data monoids, even finite ones, as shown in the following example, due to Szymon Toruńczyk. Let $\mathbb{Z}_2$ be the two element group, with additive notation. Let $h : \Gamma_\emptyset \to \mathbb{Z}_2$ assign 0 to even permutations and 1 to odd permutations. The monoid M is also $\mathbb{Z}_2$; the action is defined by $\tau m = m + h(\tau)$. The finite support axiom fails, since odd permutations can involve any pair of data values.

Nevertheless, all data monoids in this paper are homomorphic images (see below) of the free data monoid, which guarantees the finite support axiom.

Congruences of data monoids. A congruence of a data monoid is an equivalence relation $\simeq$ on the underlying monoid that is compatible with concatenation:

$$m_1 \simeq n_1 \text{ and } m_2 \simeq n_2 \qquad \text{implies} \qquad m_1 n_1 \simeq m_2 n_2 \tag{1}$$

and which is also compatible with every data renaming $\tau \in \Gamma_C$:

$$m \simeq n \qquad \text{implies} \qquad \tau m \simeq \tau n \tag{2}$$

Note that the notion of congruence depends implicitly on the set C of constants, since the set of constants indicates which functions τ are data renamings. It might be the case that a relation is a congruence for a set of constants C, but it is no longer a congruence for a smaller set of constants $E \subsetneq C$.

▶ **Lemma 1.** *For a data language $L \subseteq D^*$ that is closed under data renamings, the syntactic equivalence $\equiv_L$ is a congruence of the free data monoid D^*.*

Proof. The syntactic equivalence satisfies (1), as it does for any language $L \subseteq D^*$, not only those closed under data renamings. We use the closure under data renamings to prove (2). Let w and w' be $\equiv_L$-equivalent data words. We need to show that for any data renaming τ, also τw and $\tau w'$ are $\equiv_L$-equivalent. In other words, we need to show that

$$u(\tau w)v \in L \qquad \Leftrightarrow \qquad u(\tau w')v \in L \qquad \text{for any } u, v \in D^*. \tag{3}$$

Since τ^{-1} induces an bijection on data words, we have

$$u(\tau w)v \in L \qquad \Leftrightarrow \qquad \tau^{-1}(u(\tau w)v) \in \tau^{-1}L.$$

Because concatenating words commutes with data renamings,

$$\tau^{-1}(u(\tau w)v) = (\tau^{-1}u)w(\tau^{-1}v).$$

By the assumption on L being closed under data renamings, $L = \tau^{-1}L$. It follows that the left side of the equivalence (3) is equivalent to

$$(\tau^{-1}u)w(\tau^{-1}v) \in L.$$

Likewise, the right side of the equivalence (3) becomes

$$(\tau^{-1}u)w'(\tau^{-1}v) \in L.$$

Both sides are equivalent, because $w \equiv_L w'$. ◀

Homomorphisms of data monoids. Suppose that M and N are data monoids, with constants C. A *data monoid homomorphism* $h : M \to N$ is defined to be a monoid homomorphism of the underlying monoids that commutes with the action of data renamings

$$h(\tau s) = \tau(h(s)) \qquad \text{for all } \tau \in \Gamma_C.$$

In the literature on nominal sets, a monoid homomorphism that satisfies the commuting condition above is called *equivariant.*

A congruence gives rise to a homomorphism, in the usual way. It follows that the function that maps a word w to its equivalence class under $\equiv_L$, is a data monoid homomorphism. This function is called the *syntactic morphism.* The target of the syntactic morphism is called the *syntactic data monoid* of L, and denoted M_L.

Note that data monoid homomorphisms preserve the finite support axiom. Therefore, all the syntactic monoids we will study, as images of the free data monoid, will satisfy the finite support axiom.

We say that a data monoid homomorphism $h : D^* \to M$ *recognizes* a data language L if membership $w \in L$ is uniquely determined by the image $h(w) \in M$. In other words, $L = h^{-1}(h(L))$. Clearly, the syntactic morphism of L recognizes L. The following lemma shows that the syntactic morphism is the "best" morphism recognizing L.

▶ **Lemma 2.** *Consider a data language $L \subseteq D^*$. Any surjective data monoid homomorphism that recognizes L can be uniquely extended to the syntactic data monoid homomorphism.*

Proof. Let $\alpha : D^* \to M$ be a surjective data monoid homomorphism that recognizes L.

We claim that $\alpha(w) = \alpha(w')$ implies $w \equiv_L w'$. If the contrary were true, we would have two words w, w' with the same image under α, and some words v, u such that exactly one of the two words $vwu, vw'u$ would belong to L. This would contradict the assumption on α recognizing L, since $vwu, vw'u$ would have the same image under α.

From the claim, it follows that for every element $m \in M$, all words in $\alpha^{-1}(m)$ are mapped by the syntactic monoid homomorphism to the same equivalence class, call it W_m. Therefore, the function $m \mapsto W_m$ extends α to the syntactic data monoid homomorphism. ◀

The following lemma justifies the use of the name "free data monoid".

▶ **Lemma 3.** *Let M be a data monoid, and let $h : D \to M$ be a function that commutes with data renamings (i.e. an equivariant function). This function can be uniquely extended to a data monoid homomorphism $[h] : D^* \to M$.*

Orbits. A syntactic data monoid usually has an infinite carrier. This is not a problem, because what really interests us in a data monoid is not the elements, but their orbits. Formally, the *orbit* of an element m of a data monoid M is the set

$$\Gamma_C \cdot m = \{\tau m : \tau \in \Gamma_C\}.$$

Note how the set of constants influences the notion of orbit.

Consider the syntactic data monoid and the syntactic data monoid homomorphism of a data language $L \subseteq D^*$. As elements of the monoid are to equivalence classes of $\equiv_L$, orbits are to equivalence classes of $\simeq_L$. More formally, for two words $w, v \in D^*$, the orbits of $h_L(w)$ and $h_L(v)$ are equal if and only if $w \simeq_L v$. Suppose that a language $L \subseteq D^*$ is closed under data renamings. If this language is recognized by a data monoid homomorphism h, then the image $h(L)$ is necessarily a union of orbits.

A data monoid is called *orbit finite* if it has finitely many distinct orbits. We are interested in languages whose syntactic data monoids are orbit finite. As far as this paper is concerned, these are the "regular" languages of data words.

This completes the definition of data monoids and their basic properties.

4 Memory

As usual, the definition of data monoids attempts to use the least amount of primitive concepts. We now show how other natural concepts can be derived from these primitives.

Consider the running example language L_{dd}. Let M_{dd} be the syntactic monoid of this language. Formally speaking, elements of M_{dd} are equivalence classes of data words. The equivalence class of a data word $d_1 \cdots d_n \not\in L_{dd}$ can be identified with the ordered first/last letter pair (d_1, d_n). Intuitively speaking, the data values d_1 and d_n are "important" for this equivalence class (monoid element), while the other data values $d_2, \ldots, d_{n-1}$ are not "important". Below we present a definition, called *memory*, which automatically extracts the "important" data values from an element of a data monoid, and uses only the primitive concepts of a data monoid. This definition is inspired by the notion of memory from [2].

Let m be an element of a data monoid. We write stab_m for the subgroup of data renamings τ that satisfy $\tau m = m$. In other words, $\text{stab}_m = \Gamma_{C \cup \{m\}}$. In the running example, when m is the equivalence class of a word $d_1 \ldots d_n \not\in L_{dd}$, then stab_m is the set of data renamings that have both d_1 and d_n as fixpoints. (When m is the equivalence class of a word $w \in L_{dd}$, or m is the identity, then stab_m contains all data renamings.) For each data value $d \in D$, we define its *data orbit with respect to* m as the following set of data values

$$\{\tau d : \tau \in \text{stab}_m\} = \text{stab}_m(d).$$

In the running example, when m is the equivalence class of $d_1 \ldots d_n \not\in L_{dd}$, the data orbit of d_1 with respect to m is $\{d_1\}$, for d_n the data orbit is $\{d_n\}$, and for any other data value it is $D - \{d_1, d_n\}$.

The following theorem gives a more precise interpretation of the finite support axiom. It says that an element does not care about what happens when data values in its unique infinite data orbit are swapped around.

▶ **Theorem 4** (Memory Theorem). *Every element m of an orbit finite data monoid has finitely many data orbits, of which exactly one is infinite. Furthermore, if $mem(m)$ is defined as the union of the finite data orbits, then stab_m contains all data renamings that are the identity on $mem(m)$, i.e. $\Gamma_{mem(m)} \subseteq \text{stab}_m$.*

Note that $mem(m)$ includes all constants, since they have one element data orbits.

The result above only refers to the set structure of a data monoid, i.e. the group action on its elements. It does not refer to the monoid structure, i.e. the multiplication operation and the neutral element. Therefore, it is a result about nominal sets, as in [5]. The Memory Theorem is a corollary of Proposition 3.4 from [5], which says that an element m of a nominal set M is supported by the intersection of all sets of data values that support it.

▶ **Corollary 5.** *For m, n in a data monoid, $mem(mn) \subseteq mem(m) \cup mem(n)$.*

Proof. Let d, e be elements outside $mem(m) \cup mem(n)$. Let τ be the data renaming that swaps d, e. By the Memory Theorem, $\tau \in \text{stab}_m \cap \text{stab}_n$. It follows that $\tau(mn) = \tau m \cdot \tau n = mn$, and therefore $\tau \in \text{stab}_{mn}$. We have shown that any two elements outside $mem(m) \cup mem(n)$ are in the same data orbit for mn. Since there is only one infinite data orbit, it follows that they are not in $mem(mn)$. ◀

5 Algebraic properties of data monoids

In this section we develop the algebraic theory of data monoids.

Local finiteness. Recall that a monoid is called *locally finite* if all of its finitely generated submonoids are finite. Much of Green theory works for locally finite monoids, which makes the following theorem important.

▶ **Theorem 6.** *Every orbit finite data monoid is locally finite.*

Preliminary work indicates that local finiteness holds under much weaker conditions than being orbit finite. Namely, for every language of data words that is defined in monadic second-order logic with order and data equality, the syntactic monoid is locally finite. This covers, for instance, languages recognized by nondeterministic register automata.

Green's relations. Suppose that M is a data monoid. Using the underlying monoid, one can define Green's relations on elements $m, n \in M$.

- $m \leq_{\mathcal{L}} n$ if $Mm \subseteq Mn$
- $m \leq_{\mathcal{R}} n$ if $mM \subseteq nM$
- $m \leq_{\mathcal{J}} n$ if $MmM \subseteq MnM$

We sometimes say that n is a *suffix* of m instead of writing $m \leq_{\mathcal{L}} n$. Likewise for *prefix* and *infix*, and the relations $\leq_{\mathcal{R}}$ and $\leq_{\mathcal{J}}$. Finally, we use the name *extension* for the converse of infix. Like in any monoid, these relations are transitive and reflexive, so one can talk about their induced equivalence relations $\sim_{\mathcal{L}}$, $\sim_{\mathcal{R}}$ and $\sim_{\mathcal{J}}$. Equivalence classes of these relations are called $\mathcal{L}$-classes, $\mathcal{R}$-classes and $\mathcal{J}$-classes. An $\mathcal{H}$-class is defined to be the intersection of an $\mathcal{L}$-class and an $\mathcal{R}$-class.

In data monoids, we add a new relation, by lifting the order $\leq_{\mathcal{J}}$ to orbits. One could do the same for $\mathcal{R}$ and $\mathcal{L}$, but in our application of data monoids to characterize of first-order logic, we only use the case for $\mathcal{J}$. We write $m \leq_{\mathcal{O}} n$ if there is some data renaming τ such that $\tau m \leq_{\mathcal{J}} n$. The letter $\mathcal{O}$ stands for orbit. In terms of ideals, $\tau(MmM) \subseteq MnM$, or equivalently $M \cdot \tau m \cdot M \subseteq MnM$, or also equivalently $m \in M \cdot \tau n \cdot M$.

▶ **Lemma 7.** *In any data monoid, the relation $\leq_{\mathcal{O}}$ is transitive.*

Proof. Suppose that $x \geq_{\mathcal{O}} y \geq_{\mathcal{O}} z$ holds. This means that z can be written as $\tau(pyq)$ and y can be written as $\sigma(mxn)$, for some data renamings τ, σ and data monoid elements p, q, m, n. Combining the two, we get

$$z = \tau(p \cdot \sigma(mxn) \cdot q) = \tau(p \cdot \sigma m \cdot x \cdot \sigma n \cdot q) \in M \cdot \tau x \cdot M.$$

◀

Of course, in an orbit finite data monoid there are finitely many $\mathcal{O}$-classes, since each is a union of orbits.

$\mathcal{J}$-classes. We present a version of the Memory Theorem for $\mathcal{J}$-classes. For a $\mathcal{J}$-class J, we define its memory as

$$mem(J) = \bigcap_{m \in J} mem(m).$$

Clearly this set is finite, as an intersection of finite sets.

▶ **Theorem 8** (Memory Theorem for $\mathcal{J}$-classes.). *Let J be a $\mathcal{J}$-class. Any data renaming τ that is the identity on $mem(J)$ satisfies $\tau J = J$.*

Quotient under a $\mathcal{J}$-class. Suppose that J is a $\mathcal{J}$-class in a data monoid. Let $\simeq_J$ be the equivalence relation on the data monoid defined by:

$$m \simeq_{/J} n \qquad \text{if } m = n \text{ or neither of } m, n \text{ is an infix of some element of } J.$$

▶ **Lemma 9.** *The relation $\simeq_{/J}$ is a congruence of data monoids, assuming the constants are $mem(J)$. Consequently, the quotient function denoted $h_{/J}$, is a data monoid homomorphism, with constants $mem(J)$.*

Orbit-equivalent $\mathcal{J}$-classes. We say that two $\mathcal{J}$-classes $J, K \subseteq M$ of a data monoid are *orbit-equivalent* if there is some data renaming τ such that $\tau J = K$.

▶ **Lemma 10.** *Orbit-equivalent $\mathcal{J}$-classes form antichains in the order $\leq_{\mathcal{J}}$.*

6 First-order logic

We test the algebra on first-order logic, yielding the main result of the paper, Theorem 11. We extend first-order logic, as described in the introduction, by a unary predicate $d(x)$ for every data value d, which selects positions with data value d. A predicate $d(x)$ is called a *data predicate*. Any formula uses a finite number of data predicates, and its language is invariant under data renamings that preserve the data values of the data predicates it uses. The formulas produced below will use data predicates only for the constants.

▶ **Theorem 11.** *Let L be a data language whose syntactic data monoid M_L is orbit finite. Then L is definable in first-order logic if and only if M_L is aperiodic.*

The implication from first-order definable to aperiodic is proved in the same way as usual, e.g. using an Ehrenfeucht-Fraïssé game. We focus on the more difficult implication. The proof is an induction, which needs a more detailed statement, as presented below.

▶ Proposition 12. Let L be a data language recognized by a data monoid homomorphism $h : D^* \to M$ into a data monoid that is orbit finite and aperiodic. For every $m \in M$, the language $h^{-1}(m)$ can be defined by a sentence φ_m of FO that uses data predicates for data values in $mem(m)$.

Note that in Theorem 11 and Proposition 12, there is an implicit set of constants $C \subseteq D$, which influences the definitions of data language, data monoid, and homomorphism.

Before we prove Proposition 12, we show how it implies Theorem 11. The language L is a union of orbits. Let $m_1, \ldots, m_n$ be chosen representatives of these orbits. We know that L is the same as the union

$$\Gamma_C \cdot m_1 \quad \cup \quad \cdots \quad \cup \quad \Gamma_C \cdot m_n.$$

Each of the languages in the finite union above can be defined in first-order logic, thanks to Proposition 12 and the following lemma.

▶ **Lemma 13.** *Suppose that L is a language defined by a formula of first-order logic with data predicates A. For any $B \subseteq A$, the language*

$$\Gamma_B L = \{\tau w : \tau \in \Gamma_B, w \in L\}$$

is defined by a formula of first-order logic with data predicates B.

We now proceed with the proof of Proposition 12. The proof is by induction on two parameters, ordered lexicographically. The first, and more important, parameter is the number of orbits in the data monoid M. The second parameter is the position of the $\mathcal{O}$-class of m in the (inverse of the) order $\leq_{\mathcal{O}}$. We begin with the $\mathcal{O}$-class of the empty word, and proceed to bigger words, the idea being that it is easier to write formulas for infixes than extensions.

The induction base is done the same way as the induction step, so we omit it, and only do the induction step. Let O be the $\mathcal{O}$-class of m. By induction assumption, we can use formulas for elements with strictly bigger $\mathcal{O}$-classes, which correspond to infixes of m. There are two cases two consider, depending on whether O is a $\mathcal{J}$-class, or not. The case when O is a $\mathcal{J}$-class is more similar to the proof for finite alphabets.

O is not a $\mathcal{J}$-class. Let $J \subsetneq O$ be the $\mathcal{J}$-class that contains m. Let us expand the set of constants from C to $mem(J)$. Consider the data monoid homomorphism

$$h_{/J} : M \to M_{/J}$$

which squashes all elements that are not infixes of J into a single element, as in Lemma 9. Recall the data monoid homomorphism h that recognizes L. It is easy to see that the composition $g = h_J \circ h$ recognizes $h^{-1}(m)$, since g maps the same elements to m as h.

▶ **Lemma 14.** *$M_{/J}$ has fewer orbits than M.*

Thanks to the above lemma, we can use the induction assumption to produce the formula for m. This finishes the case when O is not a $\mathcal{J}$-class.

O is a $\mathcal{J}$-class. For $n \in M$, define K_n to be the set of data words that have image n under h, but all of their proper infixes have image in a strictly bigger $\mathcal{O}$-class than O. Using the induction assumption, we prove the following lemma.

▶ **Lemma 15.** *For each $n \in M$, the language K_n can be defined by a formula of first-order logic with data predicates $mem(n)$.*

▶ **Lemma 16.** *There are finitely many elements $m_1, \dots, m_k$ and a subset $B \subseteq mem(m)$ such that for every $n \in M$,*

$$n \sim_{\mathcal{R}} m \qquad \textit{iff} \qquad n \in \Gamma_B\{m_1, \dots, m_k\}.$$

By combining Lemmas 15, 13 and 16, we get the following lemma.

▶ **Lemma 17.** *The infinite union of languages K_n, ranging over $n \sim_{\mathcal{R}} m$, can be defined by a formula of first-order logic with data predicates $mem(m)$.*

Using Lemma 17 and its symmetric variant for $\mathcal{L}$-classes, we can write a formula of first-order logic, with data predicates $mem(m)$, which describes the set, call it L_m, of elements that have a prefix in the $\mathcal{R}$-class of m, and a suffix in the $\mathcal{L}$-class of m:

$$L_m = \bigcup_{\substack{n \\ n \sim_{\mathcal{R}} m}} K_n D^* \quad \cap \quad \bigcup_{\substack{n \\ n \sim_{\mathcal{L}} m}} D^* K_n.$$

If a word w belongs to L_m, then its image $n = h(w)$ satisfies $n \leq_{\mathcal{R}} m$, likewise for $\leq_{\mathcal{L}}$. By the following lemma, if we additionally assume that $n \sim_{\mathcal{J}} m$, then also $n \sim_{\mathcal{L}} m$ and $n \sim_{\mathcal{R}} m$, and hence $n \sim_{\mathcal{H}} m$.

▶ **Lemma 18.** *In a locally finite monoid, $m \sim_{\mathcal{J}} n$ and $m \leq_{\mathcal{R}} n$ imply $m \sim_{\mathcal{R}} n$. Likewise for $\mathcal{L}$.*

In other words, if a word belongs to L_m and has the same $\mathcal{J}$-class as m, then its image is in the $\mathcal{H}$-class of m. In a locally finite aperiodic monoid, this means that its image is simply m:

▶ **Lemma 19.** *In a locally finite aperiodic monoid, all $\mathcal{H}$-classes are singletons.*

Above we have shown that a word in L_m has image m, using the assumption that its image is in the same $\mathcal{J}$-class as m. Therefore, a word has image m if and only if it belongs to language L_m, which is definable in first-order logic, and its $\mathcal{J}$-class is O. Let K be the set of data words that have an infix in O, but are not in O. As argued before, a word has image m if and only if it belongs to the difference $L_m - K$. Therefore, to conclude we only need the following lemma.

▶ **Lemma 20.** *The language K can be defined in first-order logic.*

Proof. When does a word w belong to K? Consider the smallest infix of w that belongs to K. Cutting off the first or last letter of that infix leads to a word in O. Therefore:

$$K = D^* \cdot \Big(\bigcup_{\substack{d \in D, n \in O \\ h(d) \cdot n \not\in O}} d \cdot h^{-1}(n) \quad \cup \quad \bigcup_{\substack{d \in D, n \in O \\ n \cdot h(d) \not\in O}} h^{-1}(n) \cdot d \Big) \cdot D^*$$

The important observation is that it does no harm to replace $h^{-1}(n)$ by the language L_n defined above. This is because L_n contains $h^{-1}(n)$ and is contained in $h^{-1}(n) \cup K$. Therefore,

$$K = D^* \cdot \Big(\bigcup_{\substack{d \in D, n \in O \\ h(d) \cdot n \not\in O}} d \cdot L_n \quad \cup \quad \bigcup_{\substack{d \in D, n \in O \\ n \cdot h(d) \not\in O}} L_n \cdot d \Big) \cdot D^*$$

We deal with the infinite union using Lemma 13. ◀

7 Further work

Characterize other logics. It is natural to extend the characterization of first-order logic to other logics. Candidates that come to mind include first-order logic with two variables, or various logics inspired by XPath, or piecewise testable languages. Also, it would be interesting to see the expressive power of languages recognized by orbit-finite data monoids. This class of languages is incomparable in expressive power to first-order logic, e.g. the first-order definable language "some data value appears twice" is not recognized by an orbit-finite data monoid. It would be nice to see a logic, maybe a variant of monadic second-order logic, with the same expressive power as orbit-finite data monoids.

More structure on the data values. In this paper, we study languages that are closed under arbitrary renamings of data values. One could be interested in weaker requirements, that give more general language classes. For instance, consider languages $L \subseteq \mathbb{R}^*$ that are closed under order-preserving renamings. Another, very interesting example, concerns languages of timed automata.

Use mechanisms more powerful than monoids. Even if we are interested in languages that are preserved under arbitrary data renamings, orbit finite data monoids have weak expressive power. Contrary to the case of finite alphabets, over infinite alphabets, (orbit finite) data monoids are strictly less expressive than the natural automaton counterpart[5]. If we only require the syntactic left-to-right automaton to be orbit finite, we get a larger class of languages. This larger class includes the language "the first letter in the word appears also on some other position", which has a syntactic monoid with infinitely many orbits. Therefore, one can ask: is it decidable if an orbit finite automaton recognizes a language that can be defined in first-order logic? We conjecture that this problem is decidable, and even that a necessary and sufficient condition is aperiodicity of the syntactic monoid (which need not be orbit finite). Aperiodicity is not, in general, the right condition for first-order logic, as witnessed by the language "words with an even number of distinct letters", which has an aperiodic syntactic monoid (sic!), but is not definable in first-order logic.

References

1 Michael Benedikt, Clemens Ley, and Gabriele Puppis. Automata vs. logics on data words. In *CSL*, pages 110–124, 2010.

2 Michael Benedikt, Clemens Ley, and Gabriele Puppis. What you must remember when processing data words. In *AMW*, 2010.

3 Patricia Bouyer, Antoine Petit, and Denis Thérien. An algebraic approach to data languages and timed languages. *Inf. Comput.*, 182(2):137–162, 2003.

4 Nissim Francez and Michael Kaminski. An algebraic characterization of deterministic regular languages over infinite alphabets. *Theor. Comput. Sci.*, 306(1-3):155–175, 2003.

5 Murdoch Gabbay and Andrew M. Pitts. A new approach to abstract syntax with variable binding. *Formal Asp. Comput.*, 13(3-5):341–363, 2002.

6 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

7 Luc Segoufin. Automata and logics for words and trees over an infinite alphabet. In *CSL*, pages 41–57, 2006.

8 Howard Straubing. *Finite Automata, Formal Languages, and Circuit Complexity.* Birkhäuser, Boston, 1994.

[5] This counterpart has the same expressive power as deterministic memory automata of Francez and Kaminski [6]

Minimum $s - t$ cut in undirected planar graphs when the source and the sink are close[*]

Haim Kaplan[1] **and Yahav Nussbaum**[1]

1 **The Blavatnik School of Computer Science,**
Tel Aviv University, 69978 Tel Aviv, Israel
`{haimk,yahav.nussbaum}@cs.tau.ac.il`

Abstract

Consider the minimum $s-t$ cut problem in an embedded undirected planar graph. Let p be the minimum number of faces that a curve from s to t passes through. If $p=1$, that is, the vertices s and t are on the boundary of the same face, then the minimum cut can be found in $O(n)$ time. For general planar graphs this cut can be found in $O(n \log n)$ time. We unify these results and give an $O(n \log p)$ time algorithm. We use cut-cycles to obtain the value of the minimum cut, and study the structure of these cycles to get an efficient algorithm.

1998 ACM Subject Classification G.2.2 Graph algorithms; F.2.2 Computations on discrete structures

Keywords and phrases planar graph; minimum cut; shortest path; cut cycle

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.117

1 Introduction

The minimum $s-t$ cut problem is a well-studied problems with applications in many fields. By the Max-Flow Min-Cut Theorem [4], the value of the minimum $s-t$ cut is the same as the value of the maximum $s-t$ flow, and a minimum cut can be easily obtained from a maximum flow.

A planar graph is a graph that has an embedding in the plane such that no pair of edges cross each other. General maximum flow algorithms can solve the maximum flow and the minimum cut problems on planar graphs with n vertices and $O(n)$ edges in $O(n^2 \log n)$ time. On the other hand, algorithms that take advantage of the structure of the planar embedding of the graph can find the minimum cut and the maximum flow in $O(n \log n)$ time (see below). The history of the maximum problem on planar graphs is surveyed in [2]. In this paper we focus on undirected planar graphs.

Itai and Shiloach [11] used the correspondence between an $s-t$ cut and a cycle in the dual planar graph (see Sect. 2) separating the dual face s^* that correspond to s from the dual face t^* that corresponds to t. Such a cycle is called a *cut-cycle*. Itai and Shiloach gave an $O(n^2 \log n)$ time algorithm for finding a minimum cut using cut-cycles in undirected planar graphs. Reif [18] improved the time bound of the algorithm to $O(n \log^2 n)$ using a divide-and-conquer approach. Frederickson [5] improved the time bound of the last algorithm to $O(n \log n)$ by providing a faster shortest paths algorithm. Hassin and Johnson [8] completed the picture by showing how to find also the maximum flow within the same time bound. The

[*] This research was partially supported by the United States - Israel Binational Science Foundation, project number 2006204.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 117–128
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

cut-cycle approach was also used by Johnson [13] to get a parallel algorithm for maximum flow in directed planar graphs in $O(\log^3 n)$ time using $O(n^4)$ processors or in $O(\log^2 n)$ time using $O(n^6)$ processors. However, the sequential time bound of [13] is not better than the time bound of previous algorithms for the problem.

The $O(n \log n)$ time bound of [5, 8, 18] is the best time bound known for undirected planar graphs. This bound was matched for directed planar graphs by the maximum flow algorithm of Borradaile and Klein [2], using a different approach. The latter algorithm was simplified by Schmidt et al. [17] and Erickson [3]. However, the asymptotic running time of these simplified versions remains $O(n \log n)$.

Consider a planar graph embedded in the plane. Let p be the minimum number of faces that a curve from s to t passes through (the curve might go through vertices and edges of G). The graph is st-planar if and only if $p = 1$. This parameter p was first introduced by Itai and Shiloach [11] who gave an $O(np \log n)$ time algorithm for maximum flow in directed planar graphs if the value of the flow is known. Johnson and Venkatesan [14] gave an $O(np \log n)$ algorithm, without knowing the value of the flow in advance. The algorithm of [14] has two bottlenecks, the first one is the computation of maximum flow in st-planar graphs, and the second is transforming a planar flow into an acyclic flow. The first bottleneck was addressed by Henzinger et al. [9] and the second by Kaplan and Nussbaum [16]. Hence, we get an $O(np)$ time algorithm for flow in planar graphs, which is faster than the $O(n \log n)$ time algorithm for $p = o(\log n)$.

There is a well-known algorithm (see for example [10, Chap. 10]) for the minimum cut problem in directed st-planar graphs using a shortest path algorithm in the dual planar graph. With the shortest path algorithm of [9] this takes $O(n)$ time. Hassin [7] extended this algorithm to a maximum flow algorithm with the same time bound.

Our main result in this paper is an $O(n \log p)$ algorithm for minimum $s - t$ cut in undirected planar graphs. This algorithm runs in $O(n)$ time when the graph is st-planar ($p = 1$), matching [7], and in $O(n \log n)$ time for general undirected planar graphs, matching [5, 8, 18]. In general, p might be $\Theta(n)$, but p is small when s and t reside on the boundaries of faces which are close to each other. Our algorithm is asymptotically faster than what was previously known for any non-constant $p = o(n^\varepsilon)$ (where $\varepsilon > 0$ is constant).

Another related topological parameter q, introduced by Frederickson [6], is the minimum number of faces required to cover all vertices of the graph. It is always true that $p = O(q)$.[1] Arikati, Chaudhuri and Zaroliagis [1] gave an $O(n + q \log q)$ algorithm for the minimum $s - t$ cut problem in directed planar graphs.

We note that Janiga and Koubek [12] claimed an $O(n \log n \log p / \log \log n)$ algorithm for finding the minimum cut-cycle in directed planar graphs. Erickson [3] states that the algorithm of [12] can be implemented in $O(n \log n)$ time. However, in Appendix A we show a flaw in this algorithm.

2 Preliminaries

Let $G = (V, E)$ be an undirected simple planar graph with vertex set V and edge set E. Let $n = |V|$. Since G is planar it follows from Euler's formula that $|E| = O(n)$. We denote an edge between the vertices u and v by (u, v). We assume that the input graph is given with a

[1] Consider a curve R from s to t, and a minimum set Q of faces that cover the vertices of G. We can assume that R crosses the boundary of faces only at vertices. If two non-consecutive vertices in R are on the boundary of the same face of Q, then we can make R shorter by routing it through this face.

fixed planar embedding, in other words G is a *plane graph*.

In the graph G there are two designated vertices, the *source* s, and the *sink* t. The *capacity function* c assigns to every edge e a non-negative capacity $c(e)$.

An $s-t$ *cut*, or a *cut* for short, is a minimal set of edges S, whose removal from the graph disconnects t from s. The value of S is the total capacity of its edges, $\sum_{e \in S} c(e)$.

A *path* Q is a sequence of edges $(e_1, e_2, \dots, e_j)$ such that $e_i = (v_i, v_{i+1})$. For $1 \leq i \leq j+1$ we say that the path Q *contains* the vertex v_i. The path Q *begins* at v_1 and *ends* at v_{j+1}. A path Q is *simple* if for every vertex v, there are at most two edges incident to v in Q. We denote by $|Q|$ the number of vertices in Q. We consider a single vertex to be a degenerate path without edges. If lengths are associated with the edges then the *length* of Q is the sum of the lengths of all the edges of Q (an edge that appears in Q multiple times contributes its length to the sum the same number of times). The *reverse of* $Q = (e_1, e_2, \dots, e_j)$ is the path $Q^r = (e_j, e_{j-1}, \dots, e_1)$. For two paths $Q = (e_1, e_2, \dots, e_j)$ and $R = (d_1, d_2, \dots, d_i)$ such that the last vertex of Q is identical to the first vertex of R, we define $Q \circ R$ to be the path $(e_1, e_2, \dots, e_j, d_1, d_2, \dots d_i)$.

A path that begins and ends at the same vertex is a *cycle*. We say that two cycles are identical, if they have the same sequence of edges, in the same cyclic order (it does not matter which vertex we pick as the first/last).

A *flower-cycle* is a cycle with a special structure defined as follows. Let Q be a simple path whose last vertex is w, let B be a simple cycle that begins and ends at w. Assume that B and Q do not share any vertex except w. Then, the cycle $C = Q \circ B \circ Q^r$ is a *flower-cycle*. We call the cycle B the *blossom* of the flower-cycle C, and we call the cycle $S = Q \circ Q^r$ the *stem* of C.

For the plane graph G, the *dual graph* G^* is defined in the following way. The vertex set of G^* is the set of faces of G. Two vertices of G^* are adjacent if and only if the boundaries of the corresponding faces share an edge. The graphs G and G^* share an embedding in the plane, such that for every vertex v of G there is a unique dual face v^* in G^* that contains v, and for every vertex x^* of G^* the face x of G contains x^*. For an edge e of G, there is a single dual edge e^* that crosses e in the shared embedding of G and G^*. The capacity of an edge e of G is interpreted in G^* as the *length* of e^*. We fix our embedding such that s^* is the infinite face of G^*.[2] (See Fig. 1).

Consider a simple cycle C in G^*. The cycle C separates the plane into two connected regions. One of the regions, which contains s^*, is *outside* C and the other is *inside* C, the edges and the vertices of C are contained in both regions. We say that an edge e or a vertex v is *strictly inside* (resp. *strictly outside*) C, if it is inside (resp. outside) C, but does not belong to C.

Let C be a simple cycle in G^* such that the face t^* is inside C. The face s^* must be strictly outside C, so the cycle separates s from t in the plane. We call such C a *cut-cycle*. The edges of G that are crossed by the edges of C form an $s-t$ cut. These edges are exactly the edges whose duals are in C. The value of the cut is the same as the length of the cut-cycle C. Therefore, we get that the value of the minimum cut is the same as the value of the shortest cut-cycle [11]. (See Fig. 1).

Let Q be any path from a vertex on the boundary of s^* to a vertex on the boundary of t^*. Let $x_1^*, \dots, x_q^*$ be the vertices of Q where x_1^* is incident to s^* and x_q^* is incident to t^*. We say that an edge (x_i^*, y^*) *emanates left* from Q at x_i^*, if when traversing Q from x_1^* to x_q^*,

[2] Most recent papers, e.g. [2, 3], fix t to be on the boundary of the infinite face of G, we choose to fix s on the boundary of the infinite face to be consistent with the previous cut-cycle algorithms [8, 11, 18].

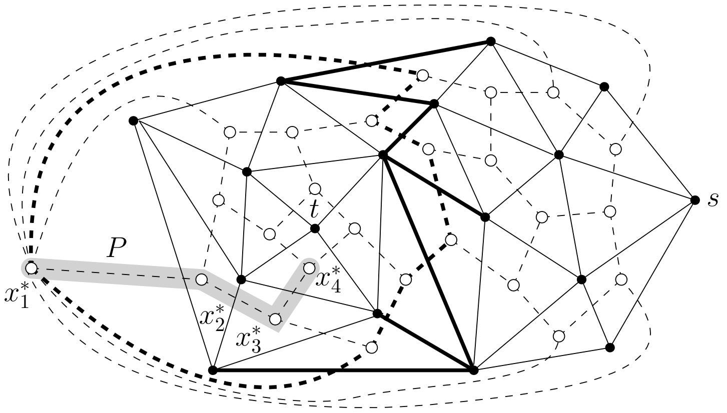


Figure 1 A planar graph and its dual. The vertices of G are *dots* and its edges are *solid.* The vertices of G^* are *circles* and its edges are *dashed.* The infinite face of G^* is s^*. The path P from s^* to t^* is *shaded.* The *bold* edges are an $s-t$ cut in G, their dual edges are a cut-cycle that contains one P-left edge incident to x_1^*.

the edge is incident to x_i^* on the left side of Q, this definition is applied to edges incident to x_1^* and x_q^* by adding two dummy vertices – x_0^* inside s^* before x_1^*, and x_{q+1}^* inside t^* after x_q^*. We call an edge *Q-left* if it emanates left from Q at exactly one of its endpoint. In other words, an edge that emanates left from Q is Q-left unless it is an edge of the form (x_i^*, x_j^*) that emanates left from Q both at x_i^* and at x_j^*. *Q-right* edges are defined similarly.

Let P be a path in G^* from a vertex on the boundary of s^* to a vertex on the boundary of t^*, with minimum number of vertices. We define $p(G)$ to be the number of vertices on P. In the introduction we defined the parameter p as the minimum number of faces that a curve from s to t passes through. This parameter is equal to $p(G)$ if the curve is not allowed to go through vertices, because then every edge of P is dual to an edge that the curve from s to t crosses.

Since we want to allow the curve to contain vertices, we change G such that in the modified graph, $\tilde{G}$, there is a curve from s to t that passes through the smallest number of faces and does not contain vertices. Furthermore, this curve in $\tilde{G}$ crosses the same number of faces as the corresponding curve in G which may contain vertices. Also, the value of the minimum $s-t$ cut in $\tilde{G}$ is the same as in G. The advantage of this transformation is that in $\tilde{G}$, p is equal to $p(\tilde{G})$ – the smallest number of vertices on a path from a vertex on s^* to a vertex on t^* in $\tilde{G}^*$.

The construction of $\tilde{G}$ is as follows. We choose a curve R from s to t passing through p faces such that R goes from a face x to a face y through a vertex v only when there is no edge common to the boundaries of x and y. If R does not contain vertices then $\tilde{G} = G$. Otherwise we split every vertex on R as follows. Consider a vertex v of G that R passes through when going from a face x to a face y. In $\tilde{G}$ we split v into two vertices v' and v'' and connect them with a new edge e that separates between x and y. Every edge that was incident to v is now incident either to v' or to v'', such that G remains planar. This transformation allows R to cross the edge e instead of the vertex v. We give e a large capacity (larger than the sum of all capacities in G), so that it does not change the minimum cut.

This transformation requires knowing the curve R. We can compute R by computing a path P' from a vertex on the boundary of s^* to a vertex on the boundary of t^* with minimum number of vertices in a graph which we construct from G^* as follows. This construction is similar to a construction of Khuller and Naor [15]. For every face v^* in G^*, we add a new vertex z^* inside the face v^* and connect z^* with edges to every vertex on the boundary of the face v^*. The new vertex z^* inside v^* allows the path P' to "jump over" the face v^* from one vertex on its boundary to another, which is equivalent to the case where the curve R passes through the vertex v of G. We also remove all the original edges from G^*. This forces R to cross edges only at incident vertices, which we can do without loss of generality. Let P' be a path with minimum number of vertices, in the graph that we constructed, from a vertex on the boundary of s^* to a vertex on the boundary of t^*. The path P' alternates between vertices of G^* and vertices that we added inside faces of G^*. For every pair of consecutive edges (x^*, z^*) and (z^*, y^*) in P', the curve R goes from the face x of G to the face y of G. If x and y share a common edge e on their boundaries in G, then R passes through e, otherwise R passes through the vertex v such that z^* is inside the face v^*.

In the rest of the paper we assume that G was preprocessed as described here (so in fact we use G to refer to $\tilde{G}$ to simplify the notation), then $p = p(G)$ is the minimum number of vertices on a path from a vertex on the boundary of s^* to a vertex on the boundary of t^*. We will denote such a path by P, we can find P in linear time using a breadth-first search on the graph G^*.

3 Finding a Minimum $s - t$ Cut

3.1 Overview

Let Π be the shortest path (path of minimum length) from a vertex incident to s^* to a vertex incident to t^*, and let $x_1^*, \ldots, x_k^*$ be the vertices on Π. Itai and Shiloach [11] observed that in an undirected planar graph, the shortest cut-cycle must cross the path Π exactly once. To exploit this observation they defined x_i^*-cycle to be a cycle containing exactly one Π-left edge and one Π-right edge, such that the Π-left edge is incident to x_i^*. Then, the minimum cut-cycle is the minimum x_i^*-cycle. Reif [18] later noticed that for every $i < j$ there is a minimum x_j^*-cycle inside a minimum x_i^*-cycle. He used this to speed up the computation of the shortest cut-cycle to $O(n \log k)$ time, using a divide-and-conquer algorithm (this is not the time bound stated by [18], but it can be obtained using techniques of [5] or the shortest-path algorithm of [9]).

If we replace Π with a path Q from s^* to t^* that is not shortest, then a cut-cycle may cross Q more than once. This makes the task of finding a shortest cut-cycle more difficult. First, there may not be a shortest cut-cycle that contains x_j^* inside a shortest cut-cycle that contains x_i^* for $i < j$, simply because x_j^* may be outside this cycle. Second, finding the shortest cut-cycle through a particular vertex is harder.

Johnson [13] showed that any cut-cycle crosses Q an odd number of times, and used this to get the parallel algorithm that we mentioned. We use the observation of Johnson that the number of crossings of a cut-cycle with Q is odd to extend the algorithm of [18] to work with any path from s^* to t^*. This way, if we take the path Q to be the path P that we defined in Sect. 2 with the minimum number of vertices from a vertex on s^* to a vertex on t^*, we get the $O(n \log p)$ time bound (recall that $p = |P|$).

Let $x_1^*, \ldots, x_p^*$ be the vertices of P, where x_1^* is incident to s^* and x_p^* is incident to t^*. First, we show how to find a shortest cut-cycle, by finding shortest simple cycles containing x_i^* that crosses P an odd number of times, for every $1 \leq i \leq p$. We characterize the structure of these cycles, and show that their structure still allows to find the shortest among them efficiently by a divide-and-conquer algorithm. Then, we show how to efficiently find the shortest cut-cycle through any particular vertex on P by computing a shortest path in a related planar graph.

3.2 Structure of shortest cut-cycles containing particular vertices

We call a cycle C an *odd-cycle* if the number of P-left edges in C is odd. The following lemma is a special case of a lemma of Johnson [13].

▶ **Lemma 1.** *Let C be a simple cycle. Then C is a cut-cycle if and only if it is an odd-cycle.*

Proof. Consider a simple cycle C and the path P. Suppose we extend P by a dummy vertex x_0^* inside s^* and by a dummy vertex x_{p+1}^* inside t^*. Assume that we walk on the plane from x_0^* to x_{p+1}^*, to the left of P and infinitesimally close to it. We start our walk outside of C (since s^* is outside of C). Each time we cross an edge of C we switch from being outside C to being inside C or vice versa. If C is a cut-cycle then x_{p+1}^* is inside C so our walk ends inside C and therefore must cross C an odd number of times. Similarly, if we cross C an odd number of times then x_{p+1}^* must be inside C and therefore C is a cut-cycle. So we conclude that C is a cut-cycle if and only if we cross C in our walk an odd number of times.

Each such crossing of the walk and C corresponds to an edge of C that emanates left from P at one of its endpoints. P-left edges emanate left from P at exactly one of their endpoints, while other edges do not emanate left from P at all, or emanate left from P at both of their endpoints. It follows that C is a cut-cycle if and only if it contains an odd number of P-left edges. ◀

We denote a *shortest odd-cycle containing the vertex x^** by $C(x^*)$. For a specific vertex x^*, the cycle $C(x^*)$ may not be simple. However, since we can decompose any cycle into simple cycles, there is a shortest odd-cycle that is simple. Moreover, any odd-cycle intersects P, so the following corollary follows from Lemma 1.

▶ **Corollary 2.** *The shortest cut-cycle is $C(x_i^*)$ for some $x_i \in P$.* ◀

Corollary 2 suggests that our definition of $C(x_i^*)$ generalizes Itai and Shiloach's definition of a minimum x_i^*-cycle. This is essential since, as we mentioned, when we replace Π by the path P, which is not a shortest path, there may be more than one P-left edge in a shortest cut-cycle.

The next lemma allows us to assume that $C(x^*)$ is a flower-cycle.

▶ **Lemma 3.** *For any vertex x^*, there is a shortest odd-cycle containing x^* that is a flower-cycle.*

Proof. The edge set of the cycle $C(x^*)$ is a union of edge sets of simple cycles, at least one of these simple cycles must be an odd-cycle. Let C be such a simple odd-cycle, and let y^* be the first vertex of C that we encounter when we traverse $C(x^*)$, starting at an occurrence of x^*.

We can decompose $C(x^*)$ into $Q \circ C \circ Q'$, where Q is a path from x^* to y^* and Q' is a path from y^* to x^*. The length of Q must be equal to the length of Q' since otherwise we can replace the longer by the reverse of the shorter and get a shorter odd-cycle through x^* (C is an odd-cycle and the sets of P-left edges of a path and its reverse are identical). Let $F = Q \circ C \circ Q^r$. The cycle F is a flower-cycle by its definition, it is also an odd-cycle, and it has the same length as $C(x^*)$. Therefore, F is a shortest odd-cycle containing x^* which is a flower-cycle. ◀

Recall that Reif [18] based his divide-and-conquer approach on the observation that inside every minimum x_i^*-cycle there is a minimum x_j^*-cycle if $i < j$. The same claim is not always true for $C(x_i^*)$ and $C(x_j^*)$. It might be possible that $C(x_i^*)$ is actually strictly inside $C(x_j^*)$. (See Fig. 2). We develop an alternative similar divide-and-conquer approach for $C(x_i^*)$.

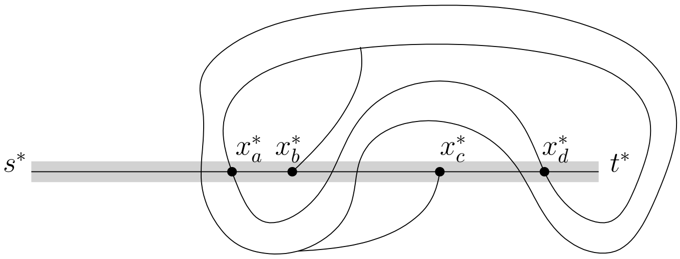


Figure 2 Structure of the flowers-cycles. $C(x_a^*)$ has an empty stem, $C(x_b^*)$ has the same blossom as $C(x_a^*)$ and stem inside the blossom, $C(x_c^*)$ has a stem outside its blossom. There are 3 P-left edges in $C(x_a^*)$ and in $C(x_c^*)$, and 5 P-left edges in $C(x_b^*)$ (the edge of the stem is counted twice). $C(x_b^*)$ is strictly inside $C(x_c^*)$ even though $b < c$. We do not have to compute any $C(x_i^*)$ for $i < d$ inside $C(x_b^*)$, even if one of them is a minimal cut-cycle we will find it for another value of i.

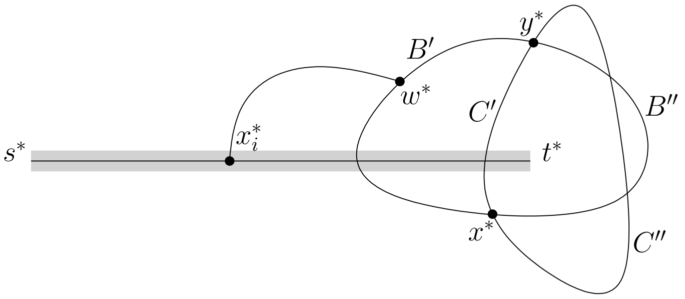


Figure 3 The shortest cut-cycle $C = C' \circ C''$ is neither inside nor outside the blossom $B = B' \circ B''$.

▶ **Lemma 4.** *Let B be the blossom of $C(x_i^*)$ for some i, and let S be the stem of $C(x_i^*)$. There is a shortest cut-cycle that is either inside B, or outside B (if B is a shortest cut-cycle then both hold).*

Proof. Assume otherwise, then for every shortest cut-cycle C, there are edges of C strictly inside B and edges of C strictly outside B.

Let w^* be the vertex common to B and S. By the minimality of $C(x_i^*)$ we may assume that $C(w^*) = B$.

Let C be a shortest cut-cycle maximizing the number of edges that it has in common with B. Let C' be a maximal subpath of C strictly inside B. Let C'' be “the rest of C” – that is, the path such that $C = C' \circ C''$. The path C' starts at a vertex x^* on B and ends at another vertex y^* on B. We split B or B^r into two parts, B' and B'', such that B' is a path from x^* to y^*, and B'' is a path from y^* to x^*, and the cycle $C' \circ B''$ is an odd-cycle. Since both B and C are odd cycles, such a decomposition of B or of B^r must exist. It follows that both $C' \circ B''$ and $B' \circ C''$ are odd-cycles. (See Fig. 3).

By our choice of C, we may assume that C' is shorter than B', as otherwise $B' \circ C''$ is not longer than C, and has more edges in common with B, contradicting the choice of C. Also, we may assume that C'' is shorter than B'', as otherwise $C' \circ B''$ is not longer than C, and has more edges in common with B, again in contradiction to the choice of C. Therefore both $C' \circ B''$ and $C'' \circ B'$ are shorter than B. The vertex w^* must be on one of the cycles $C' \circ B''$ or $B' \circ C''$, contradicting the minimality of $B = C(w^*)$. Hence the lemma follows. ◀

▶ **Lemma 5.** *Let B be the blossom of $C(x_i^*)$ for some i, and let S be the stem of $C(x_i^*)$. If B is not a shortest cut-cycle, then any shortest cut-cycle does not contain any edge incident* *to a vertex of S.*

Proof. Assume that B is not a shortest cut-cycle, and let C be a shortest cut-cycle, such that there is a vertex x^* that is common to S and C. Let Q be the shortest path from x_i^* to x^*. By its definition, the path Q is not longer than half of the cycle S.

The cycle $Q \circ C \circ Q^r$ is an odd-cycle since C is an odd cycle. The cycle C is shorter than B, and the cycle $Q \circ Q^r$ is not longer than S, contradicting the minimality of $C(x_i^*)$. ◀

3.3 Divide-and-conquer algorithm

Lemma 4 gives a method for dividing the graph, in order to find a shortest cut-cycle. Consider $C(x_i^*)$ with blossom B and stem S. If B is not a shortest cut-cycle then there is a shortest cut-cycle C, such that C is either inside B or outside B. Thus, we can divide the graph into two parts, G_{in} which is the part inside B, and G_{out} which is the part outside B, and search in each of them separately. We also discard S from the subgraph containing it, which we can do by Lemma 5. This will help us to bound the depth of the recursion.

It is simpler to describe how to obtain G^*_{in} and G^*_{out} from G^*, so we do this first. We start by putting into G^*_{in} every edge and vertex of G^* that is inside B. The part of the plane strictly outside B becomes the infinite face s^*_{in}, we also set $t^*_{\text{in}} = t^*$. The graph G^*_{out} initially contains every edge and vertex of G^* that is outside B. (By our definitions of “outside” and “inside” there is a copy of B in both graphs.) In G^*_{out} we set $s^*_{\text{out}} = s^*$ and the part of the plane inside B becomes a single face which we denote by t^*_{out}.

Assume that S is not empty. Since there is a single vertex w^* that is common to S and B, S is either entirely inside B or outside B. Assume that S is outside B. According to Lemma 5 we may assume that if there is a vertex of S on a shortest cut-cycle, then it is w^*. This happens only when B is a shortest cut-cycle, in this case B is contained also in G^*_{in}. Thus, we can remove the vertices of S and all the edges adjacent to them from G^*_{out} without losing the shortest cut-cycle. Symmetrically, if S is inside B, then we remove the vertices of S and their adjacent edges from G^*_{in}. This completes the definition of G^*_{in} and G^*_{out}.

The effect of this construction on the primal graph is as follows. Consider the common embedding of G and G^*. The graph G_{in} contains all the vertices inside B and all edges with both endpoints inside B. Similarly, G_{out} contains the vertices outside B and edges with both endpoints outside B. Edges of G whose duals are in B are the edges with one endpoint outside B and one endpoint inside B. We put a copy of these edges in both graphs as follows. We add a vertex s_{in} to G_{in}, which would be the source of G_{in}, and for every edge $e = (u, v)$ such that $e^* \in B$ with v inside B, we put the edge (s_{in}, v) in G_{in} with the same capacity as of e. Similarly, we add a vertex t_{out} to G_{out}, which would be the sink of G_{out}, and for every edge $e = (u, v)$ such that $e^* \in B$ with u outside B, we put the edge (u, t_{out}) in G_{out} with the same capacity as of e. We set $t_{\text{in}} = t$ and $s_{\text{out}} = s$.

If S is not empty, then we contract every edge $e = (u, v)$ such that e^* is incident to a vertex of S, in the graph whose dual contains S. That is u and v become a single vertex, and the incidence lists are concatenated (without e) in the appropriate cyclic order. Note that the edges that we contract include all the edges on faces of G that correspond to the vertices of S in G^*. The contraction eliminates all these faces.

It is possible that the new graphs G_{in} and G_{out} are not simple. We replace a set of multiple edges of the form (s_{in}, v) or (u, t_{out}) by a single edge whose capacity is the sum of all capacities of the multiple edges. We do so to ensure that G_{in} and G_{out} remain simple. Two parallel edges e and d create a face x between them. In the dual graph, x^* is a vertex adjacent only to e^* and d^*. The effect in the dual graph of merging e and d is the removal of

x^*, and replacement of e^* and d^* by a single edge whose length is the sum of lengths of both dual edges.

Note that every vertex of G has a single copy, either in G_{in} or in G_{out}, while edges and faces of G whose duals are in B might have copies in both graphs. When we construct G_{in} and G_{out} from G, every edge of G is mapped to a single edge of G_{in} or G_{out}. However, a single edge of G_{in} or G_{out} might be mapped to more than one edge of G (due to merge of parallel edges). When we return a cut in G_{in} or G_{out} as an answer to the minimum cut problem on G, we replace every edge of G_{in} or G_{out} with all the edges of G that were mapped to it.

Now we have all the definitions required to present the divide-and-conquer algorithm for finding a minimum $s-t$ cut:

1. Find P, the path with minimum number of vertices from a vertex on the face s^* to a vertex on the face t^*. Let $p = p(G) = |P|$.
2. If $1 \leq p \leq 2$, find $C(x_i^*)$ for every $1 \leq i \leq p$, and return the shortest.
3. Otherwise, let $i = \lfloor p/2 \rfloor + 1$.
4. Find $C(x_i^*)$.
5. Construct G_{in}, G_{out} and apply the algorithm recursively to them.
6. Return the smaller between the minimum $s_{\text{in}} - t_{\text{in}}$ cut in G_{in} and the minimum $s_{\text{out}} - t_{\text{out}}$ cut in G_{out} that were computed in the previous step.

We already pointed out two differences between our algorithm and these of Reif [18] and Hassin and Johnson [8], namely using the path P instead of the shortest path Π and finding a cut-cycle that is a $C(x_i^*)$ cycle instead of a minimum x_i^*-cycle.[3] Another difference is that we do not compute $C(x_i^*)$ for every x_i^* in the original path P from s^* to t^*. Since $C(x_i^*)$ may cross P multiple number of times it is possible, for example, that for some $j > i$, x_j^* is on the boundary of s_{in}^*, and so we do not need to apply our algorithm in G_{in} for $x_{i'}^*$ such that $i' < j$ (see Fig. 2). A symmetric claim is true for G_{out}. For this reason, we compute the path P in the first step of each recursive call. This is easy to do in time linear in the size of the input graph, by using breadth-first search on the dual graph.

The correctness of our algorithm follows from Lemma 4 and Lemma 5. In order to get the desired $O(n \log p)$ time bound, we show that the depth of the recursion is $\lceil \log p \rceil + 1$ and that it is possible to find $C(x_i^*)$ in $O(n)$ time.

To bound the depth of the recursion we show that $p(G_{\text{in}}), p(G_{\text{out}}) \leq \lfloor p/2 \rfloor + 1$.[4]

First, assume that S is empty, that is, $x_i^* \in B$. The copy of the vertex x_i^* in the graph G_{in}^* is on the boundary of s_{in}^*. The subpath $(x_i^*, x_{i+1}^*, \ldots, x_p^*)$ of P is not necessarily in G_{in}^*, but G_{in}^* must contain a suffix of this subpath that starts with some x_j^*, $j \geq i$, that is on the boundary of s_{in}^*. This implies that $p(G_{\text{in}}) \leq \lfloor p/2 \rfloor + 1$. Similarly, the copy of the vertex x_i^* in G_{out}^* is on the boundary of t_{out}^* so there is a prefix $(x_1^*, \ldots, x_{j'-1}^*, x_{j'}^*)$, $j' \leq i$ of P, such that $x_{j'}^*$ is on the boundary of t_{out}^* in G_{out}^*. This shows that $p(G_{\text{out}}) \leq \lfloor p/2 \rfloor + 1$.

Now assume that S is not empty and that it is outside B. Let w^* be the vertex common to S and B. Since B is an odd-cycle it must contain a P-left edge. Since x_i^* is outside B, at

[3] Another minor change from the algorithm of [18] is in the base of the recursion (Step 2), this correction was suggested by [8].

[4] Consider a binary representation b of p. We obtain a binary representation of an upper bound on the new value of p following a recursive call, by shifting b one position to the right and adding one to the result. It follows that the number of bits in the representation of the upper bound decreases by one every step but can increase by one at most once, so the depth of the recursion is at most the number of bits in b plus one.

least one P-left edge of B is incident to x_j^* for some $j \geq i$, so the proof that $p(G_{\text{in}}) \leq \lfloor p/2 \rfloor + 1$ does not change. We removed the vertices of S and their incident edges from G_{out}^*. Before this removal, w^* was on the boundary of t_{out}^*, so after the removal, every vertex that was adjacent to a vertex of S, is on the boundary of t_{out}^*. The vertex x_i^* is in S, so there must be a vertex $x_{i'}^*$ with $i' < i$ that was adjacent to vertex in S, and is now on the boundary of t_{out}^*. We get that there is a prefix $(x_1^*, \ldots, x_{j'-1}^*, x_{j'}^*)$, $j' \leq i'$, of P in G_{out}^* which shows that $p(G_{\text{out}}) \leq \lfloor p/2 \rfloor$. The proof for the case where S is inside B is symmetric.

We conclude that the depth of the recursion is at most $\lceil \log p \rceil + 1$.

3.4 Finding a shortest odd-cycle containing a particular vertex

Now we show how to find $C(x_i^*)$, a shortest odd-cycle that is a flower-cycle containing a specific vertex x_i^* of P. We do so in $O(n)$ time using the following construction.

We create a new planar graph H that contains two modified copies of G^* as follows. For every vertex x^* in G^* we create two copies in H, x^0 and x^1. For every edge (x^*, y^*) that is not a P-left edge we create two copies (x^0, y^0) and (x^1, y^1) in H. Last, for every P-left edge (x_j^*, y^*) we create two copies (x_j^0, y^1) and (x_j^1, y^0). We denote the set that contains the vertices x^0, the edges (x^0, y^0) and the edges (x_j^0, y^1) by H^0, and the set of other vertices and edges by H^1. A path from x^* to y^* in G^* corresponds to a path from x^0 to y^0 or a path from x^0 to y^1 in H. We denote the image in H of a path Q in G^* by $h(Q)$.

Let Q be a cycle in G^*, fix x^* to be the first vertex of Q. The image $h(Q)$ begins with x^0, which is in H^0. Assume that we traverse $h(Q)$, starting at x^0. Every time that $h(Q)$ goes through an image of a P-left edge, it jumps from H^0 to H^1 or vice versa. Therefore, Q is an odd-cycle if and only if $h(Q)$ ends at x^1.

For a vertex x_i^* of P, we find a shortest odd-cycle through x_i^* by finding a shortest path R from x_i^0 to x_i^1 in H. The cycle $h^{-1}(R)$ is a shortest odd-cycle containing x_i^*.

Although by Lemma 3 there is a shortest odd cycle through x_i^* which is a flower-cycle the cycle $h^{-1}(R)$ may not be a flower-cycle. We can convert it to a flower-cycle as suggested by the proof of Lemma 3. Let y^* the first vertex that repeats twice on $h^{-1}(R)$, and let C be the path in $h^{-1}(R)$ between the first two occurrences of y^*. If C is not and odd-cycle (this may happen only if the length of C is 0), then remove the edges of C from $h^{-1}(R)$ and repeat the process until a simple odd-cycle is found. Let Q be the prefix of $h^{-1}(R)$ that ends at y^*. Finally, let $C(x_i^*) = Q \circ C \circ Q^r$.

The construction of H takes $O(n)$ time, and finding R takes $O(n)$ time using the algorithm of Henzinger et al. [9]. Replacing $h^{-1}(R)$ with a flower-cycle takes $O(n)$ time as well. We conclude that a single recursive application of our algorithm is linear in the size of the graph it works on.

3.5 Running time

The running time analysis for our main algorithm is similar to that of Reif [18] or Hassin and Johnson [8], we use here the one of [8]. As we showed, the depth of the recursion tree is at most $\lceil \log p \rceil + 1$. In each recursive call, when we split G into G_{in} and G_{out}, we add two vertices to the graphs. Therefore, at the ℓ^{th} level of the recursion we have at most $n + 2^\ell$ vertices. The time bound at each level of the recursion is linear in the number of vertices at the level so the total running time of our algorithm is $O\left(\sum_{\ell=0}^{\lceil \log p \rceil} \left(n + 2^\ell\right)\right) = O(n \log p)$.

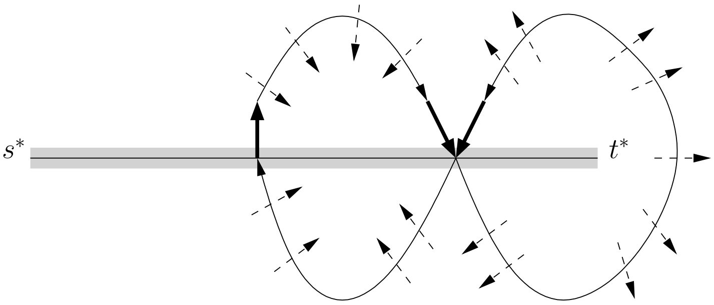


Figure 4 Counterexample to the algorithm of [12] (see [13, Fig. 7]). The directed cycle contains three P-left arcs (*bold*), such that the arc farthest from t^* is oriented away from P. However, the cycle is not simple and it wraps around t^* in the wrong direction. The direction of the primal arcs that correspond to the cycle is indicated with *dashed arrows*.

A The Algorithm of [12]

Janiga and Koubek [12] presented an $O(n \log n \log p / \log \log n)$ algorithm for the minimum $s - t$ cut problem in directed planar graphs, based on the cut-cycles approach. Erickson [3] states that this algorithm can be implemented in $O(n \log n)$ time. In this appendix we show that there is a mistake in the algorithm of [12].

To deal with directed graphs we have to extend the definitions from Sect. 2. First, in a directed graph, two anti-parallel arcs, $(u \to v)$ and $(v \to u)$ may have different capacities. Second, the dual graph G^* is also directed. The dual of an arc $d = (u \to v)$ is the arc in G^* which is directed from the dual vertex of the face on the *right side* of d to the dual vertex of the face on the *left side* of d.[5] The path P that the algorithm of [12] uses is a *directed path* with minimum number of vertices from a vertex on the boundary of s^* to a vertex on the boundary of t^*.

In the directed graph G^*, a cut-cycle is dual to a cut in G, if and only if it is oriented *clockwise* around t^* in the shared embedding of G and G^* in the plane [13].

Janiga and Koubek [12, Sect. 3] look for the minimum cut by computing the shortest cycle in G^* that its P-left arc farthest from t^* is oriented away from P (that is, the P-left arc which is adjacent to x_i^* for the minimum i is oriented $(x_i^* \to y)$), and that crosses P an odd number of times. If this shortest cycle is *simple*, then it is a cut-cycle oriented clockwise around t^* [13] (the proof is similar to Lemma 1). However, it is possible that the shortest cycle that fulfills these requirements is not simple. In this case, the cycle may be oriented counterclockwise around t^*, and therefore it would not be dual to a cut. Figure 4 shows an example of such case, which was given by Johnson [13, Fig. 7]. Since algorithm *Cycle2* of [12, p. 43] fails to check whether the path it finds it simple or not, the algorithm of [12] will not find a correct solution in this case.

References

1 Arikati, S.R., Chaudhuri, S., Zaroliagis, C.D.: All-pairs min-cut in sparse networks. *J. Algorithms* 29, 82–100 (1998)

2 Borradaile, G., Klein, P.: An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *J. ACM* 56, 1–30 (2009)

[5] Some papers, e.g. [2, 3], use the opposite orientation.

3 Erickson, J.: Maximum flows and parametric shortest paths in planar graphs. In: *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 794–804 (2010)

4 Ford, L.R., Fulkerson, D.R.: *Flows in Networks.* Princeton University Press, New Jersey (1962)

5 Frederickson, G.N.: Fast algorithms for shortest paths in planar graphs, with applications. *SIAM J. Comput.* 16, 1004–1022 (1987)

6 Frederickson, G.N.: Using cellular graph embeddings in solving all pairs shortest path problems. *J. Algorithms* 19, 45–85 (1995)

7 Hassin, R.: Maximum flow in (s,t) planar networks. *Information Processing Letters* 13, 107 (1981)

8 Hassin, R., Johnson, D.B.: An $O(n \log^2 n)$ algorithm for maximum flow in undirected planar networks. *SIAM J. Comput.* 14, 612–624 (1985)

9 Henzinger, M.R., Klein, P., Rao, S., Subramania, S.: Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.* 55, 3–23 (1997)

10 Hu, T.C.: *Integer Programming and Network Flows.* Addison-Wesley, MA (1969)

11 Itai, A., Shiloach, Y.: Maximum flow in planar networks. *SIAM J. Comput.* 8, 135–150 (1979)

12 Janiga, L., Koubek, V.: Minimum cut in directed planar networks. *Kybernetika* 28, 37–49 (1992)

13 Johnson, D.B.: Parallel algorithms for minimum cuts and maximum flows in planar networks. *J. ACM* 34, 950–967 (1987)

14 Johnson, D.B., Venkatesan, S.M.: Partition of Planar Flow Networks. In: *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pp. 259–264 (1983)

15 Khuller, S., Naor, J.: Flow in planar graphs with vertex capacities. *Algorithmica 11*, 200–225 (1994)

16 Kaplan, H., Nussbaum, Y.: Maximum flow in directed planar graphs with vertex capacities. *Algorithmica*, *in press*

17 Schmidt, F.R., Toppe, E., Cremers, D.: Efficient planar graph cuts with applications in Computer Vision. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 351–356 (2009)

18 Reif, J.H.: Minimum *s*-*t* cut of a planar undirected network in $O(n \log^2(n))$ time. *SIAM J. Comput.* 12, 71–81 (1983)

Towards Duality of Multicommodity Multiroute Cuts and Flows: Multilevel Ball-Growing*

Petr Kolman[1] and Christian Scheideler[2]

1 Faculty of Mathematics and Physics, Charles University in Prague
Malostranské nám. 25, 118 00 Prague, Czech republic
kolman@kam.mff.cuni.cz

2 Dept. of Computer Science, University of Paderborn
Fürstenallee 11, 33102 Paderborn, Germany
scheideler@upb.de

Abstract

An *elementary h-route flow*, for an integer $h \geq 1$, is a set of h edge-disjoint paths between a source and a sink, each path carrying a unit of flow, and an *h-route flow* is a non-negative linear combination of elementary h-route flows. An *h-route cut* is a set of edges whose removal decreases the maximum h-route flow between a given source-sink pair (or between every source-sink pair in the multicommodity setting) to zero. The main result of this paper is an approximate duality theorem for multicommodity h-route cuts and flows, for $h \leq 3$: The size of a minimum h-route cut is at least f/h and at most $O(\log^3 k \cdot f)$ where f is the size of the maximum h-route flow and k is the number of commodities. The main step towards the proof of this duality is the design and analysis of a polynomial-time approximation algorithm for the minimum h-route cut problem for $h = 3$ that has an approximation ratio of $O(\log^3 k)$. Previously, polylogarithmic approximation was known only for h-route cuts for $h \leq 2$. A key ingredient of our algorithm is a novel rounding technique that we call multilevel ball-growing. Though the proof of the duality relies on this algorithm, it is not a straightforward corollary of it as in the case of classical multicommodity flows and cuts. Similar results are shown also for the sparsest multiroute cut problem.

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

Keywords and phrases Multicommodity flow; Multiroute flow; Cuts; Duality

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.129

1 Introduction

The celebrated maximum-flow minimum-cut theorem of Ford and Fulkerson [5] is among the most important results in combinatorial optimization. Its importance has influenced the search for various generalizations. In the *maximum multicommodity flow problem* the goal is to maximize the sum of flows between given source-sink pairs subject to capacity constraints. In the dual problem, namely in the *minimum multicut problem*, the objective is to find a subset of edges of minimum total capacity whose removal disconnects each of the given source-sink pairs. Though an exact duality theorem does not apply to these two problems, Garg et al. [6], building on an earlier work of Leighton and Rao [10] and of others, proved an approximate max-flow min-cut theorem; the approximation factor is logarithmic

* This research was partially supported by the grants GA ČR 201/09/0197, 1M0021620808 (ITI), and DFG SCHE 1592/1-1.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 129–140
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

in the number of commodities and is asymptotically optimal. The results are proved using the ball-growing (also known as region-growing) technique that was introduced in the paper of Leighton and Rao.

Multi-route flows and multi-route cuts generalize in a natural way the concept of classical flows and cuts in graphs. An *elementary h-route flow*, for an integer $h \geq 1$, is a set of h edge-disjoint paths between a source and a sink, each path carrying a unit of flow, and an *h-route flow* [8, 1] is a non-negative linear combination of elementary h-route flows. An *h-route cut* is a set of edges whose removal disconnects a given source-sink pair with respect to h-route flows (in the multicommodity setting, it disconnects every source-sink pair). In other words, an h-route cut is a set of edges whose removal decreases the edge-connectivity of a given source-sink pair (or of every given source-sink pair) below h. Note that for $h = 1$, h-route flows and h-route cuts correspond to the classical flows and cuts.

1.1 Our results and techniques

The main result of this paper is an approximate duality theorem for multicommodity h-route cuts and flows for $h \leq 3$. In particular, we prove an upper bound of $O(\log^3 k \cdot f)$ on the size of a minimum h-route cut where f is the size of a maximum h-route flow and k is the number of source-sink pairs (or *commodities*); trivially, f/h is a lower bound.

A major step towards the proof of the duality in this paper is the design and analysis of an approximation algorithm for the minimum 3-route cut problem. The approximation ratio of our algorithm is $O(\log^3 k)$. This provides a partial answer to open problems of several papers (Bruhn et al. [3], Chekuri and Khanna [4] and Barman and Chawla [2]). The 3-route cut problem is more complicated than the 1-route and 2-route cut problems: while 1-route and 2-route cuts separate the graph into independent parts, h-route cuts do not have this property for $h > 2$. For example, when providing a 2-route cut C for the commodity (s_1, t_1) that partitions the graph into the node sets S_1 and T_1 with at most one remaining edge between them, then the commodities that have both nodes in S_1 or both in T_1 can be treated independently because no simple path can connect two nodes in S_1 (resp. T_1) via a path through T_1 (resp. S_1). This is not the case for 3-route cuts where a simple path between two nodes in S_1 may very well pass through T_1. A key ingredient to handle this problem in our paper is a novel rounding technique, called multilevel ball-growing, a generalization of the well-known ball growing argument that makes it possible to control the dependencies between parts of the graph that are separated by 3-route cuts.

Though the proof of the duality relies on the approximation algorithm, it is not a straightforward corollary of it as is the case for classical multicommodity flows and cuts. For the duality proof we show a tight relationship between two different linear relaxations [4, 2] of the h-route cut problem.

1.2 Other related results

The concept of multi-route flows was introduced by Kishimoto and Takeuchi [8]. As far as we know, the problem of a minimum h-route cut, for $h > 1$, was first considered by Bruhn et al. [3] in a paper dealing primarily with single source multi-route flows on graphs with uniform capacities. In this particular setting they established an approximate max-flow min-cut theorem and, as a corollary, described a $(2h - 2)$-approximation algorithm for the minimum h-route cut problem, for any $h > 1$.

For graphs with non-uniform capacities, the first non-trivial approximation for multi-route cuts was given by Chekuri and Khanna [4]. They dealt with the special case of $h = 2$ and provided an $O(\log^2 n \log k)$-approximation for the 2-route cut problem where n is the number of vertices in G. As their algorithm is based on an LP relaxation that is dual to the LP for the maximum 2-route flow problem, an implicit corollary of their result is an approximate duality of 2-route flows and 2-route cuts. The approximation factor for 2-route cuts was recently improved by Barman and Chawla [2] who described an $O(\log^2 k)$-approximation for the 2-route cut problem. Their algorithm is based on a different linear programming relaxation that allows them to extend the classical (discrete) ball-growing (or region-growing) technique (cf. [10, 6, 11]) to 2-route cuts. In a subsequent work [9], using a combination of the multilevel ball-growing technique and other arguments, we proved an approximate duality theorem for multicommodity h-route cuts and flows for *any* h, on uniform capacity networks. A challenging open problem is to prove an analogous result for networks with general capacities.

2 Minimum h-Route Cut Problem

Suppose that we are given a minimum h-route cut problem for the graph $G = (V, E)$ with edge capacities $c : E \to \mathbb{R}_+$ and with commodities $(s_1, t_1), \ldots, (s_k, t_k)$. If $F \subseteq E$ is an h-route cut for the instance, then for every commodity there exists a set F_i of at most $h - 1$ edges such that $F \cup F_i$ is a classical cut for the commodity i. With this observation, the integer LP for the minimum h-route cut problem can be stated as follows (by $\mathcal{P}_i$ we denote the set of all edge-simple paths in G between s_i and t_i):

$$\begin{aligned} &\min \sum_{e \in E} c(e)x(e) && && \qquad (1)\\ \sum_{e \in p} (x(e) + x_i(e)) &\geq 1 && \forall i \in [k], p \in \mathcal{P}_i \\ \sum_{e \in E} x_i(e) &\leq h - 1 && \forall i \in [k] \\ x(e) &\in \{0, 1\} && \forall e \in E \\ x_i(e) &\in \{0, 1\} && \forall i \in [k], \forall e \in E \end{aligned}$$

In order to find a good approximate solution for this ILP, we will look at its LP relaxation where $x(e) \in \{0, 1\}$ is replaced by $x(e) \geq 0$ and $x_i(e) \in \{0, 1\}$ is replaced by $x_i(e) \geq 0$. In the following, let the x- and x_i-values represent an optimal solution of this LP relaxation and let $\phi = \sum_{e \in E} c(e)x(e)$. Our goal is to round these values to an integral solution with cost at most $O(\phi \log^3 k)$ for $h = 3$. For this we will use a novel rounding technique that we call *multilevel ball-growing*. At the heart of this (as well as the classical ball growing) technique is the following lemma from elementary calculus.

▶ **Lemma 1.** *Let $[l_1, r_1], [l_2, r_2], \ldots, [l_z, r_z]$ be internally disjoint intervals of real numbers such that $l_1 < l_2 < \cdots < l_z$ and let $\mathcal{R} = \bigcup_{i=1}^{z} [l_i, r_i]$. Assume that the following holds:*

- *f is a nondecreasing function on $\mathcal{R}$ and $f(l_1) > 0$,*
- *f is differentiable on $\mathcal{R}$, except for finitely many points,*
- *g is a function on $\mathcal{R}$ such that $\forall r \in \mathcal{R}$, $g(r) \leq f'(r)$, except for finitely many points.*

Let $\gamma = f(r_z)/f(l_1)$. Then there exists $r \in \mathcal{R}$ such that $g(r) \leq \frac{1}{|\mathcal{R}|} \log \gamma \cdot f(r)$.

Proof. Assume, by contradiction, that for every $r \in \mathcal{R}$ we have $g(r) > \frac{1}{|\mathcal{R}|} \log \gamma \cdot f(r)$. Then

$$\log \gamma \leq \int_{r \in \mathcal{R}} \frac{1}{|\mathcal{R}|} \log \gamma \, \mathrm{d}r < \int_{r \in \mathcal{R}} \frac{g(r)}{f(r)} \, \mathrm{d}r \leq \int_{r \in \mathcal{R}} \frac{f'(r)}{f(r)} \, \mathrm{d}r \leq \log \frac{f(r_z)}{f(l_1)} = \log \gamma \;,$$

a contradiction. ◀

3 Single Source

Our algorithm for 3-route cuts works in iterations, and in iteration i some 3-route cut is found for some commodity i that does not yet have a 3-route cut. These 3-route cuts are added up to some final cut $F \subseteq E$. Our goal is to make sure that $c(F) = O(\log^3 k \sum_e c(e) \cdot x(e)) = O(\phi \log^3 k)$. We start with some basic notation for iteration i.

We define $d_y(u)$ as the length of the shortest path from t_i to the node u with respect to the length function $y : E \to \mathbb{R}_{\geq 0}$. For the definitions of the δ-sets in iteration i (see below) we view every edge $uv \in E$ as a segment consisting of two parts: an x-part of length $x(uv)$ followed (on the way from t_i) by an x_i-part of length $x_i(uv)$. Certainly, $x(uv) + x_i(uv) \geq |d_{x+x_i}(v) - d_{x+x_i}(u)|$ for every edge $uv \in E$ but for the definition of the δ-sets below it will be convenient to assume equality between the two quantities. To ensure the equality, we perform a minor temporary modification of the x and x_i values: if $x(uv) \leq |d_{x+x_i}(v) - d_{x+x_i}(u)|$ then we reduce $x_i(uv)$ to $|d_{x+x_i}(v) - d_{x+x_i}(u)| - x(uv)$, otherwise we reduce $x(uv)$ to $|d_{x+x_i}(v) - d_{x+x_i}(u)|$ and set $x_i(uv) = 0$. These adjustments are *only* valid for the following definitions.

In iteration i, for any $r \in [0, 1]$ we define

$$\begin{aligned}
B(r) &= \{u \in V \mid d_{x+x_i}(u) \leq r\} \\
\delta(r) &= \{uv \in E \mid d_{x+x_i}(u) \leq r < d_{x+x_i}(v)\} \\
\delta_x(r) &= \{uv \in \delta(r) \mid d_{x+x_i}(u) \leq r \leq d_{x+x_i}(u) + x(uv)\} \\
\delta_{x_i}(r) &= \{uv \in \delta(r) \mid d_{x+x_i}(v) - x_i(uv) < r \leq d_{x+x_i}(v)\}
\end{aligned}$$

In words, the set $B(r)$, called a *ball* (or region) with center at t_i and radius r, is the set of nodes at distance at most r from t_i (with respect to $x + x_i$); $\delta(r)$ is the set of edges in the cut between $B(r)$ and $V \setminus B(r)$, $\delta_x(r)$ is the subset of edges from the cut $\delta(r)$ that are cut in their x-part, and $\delta_{x_i}(r)$ are those from $\delta(r)$ that are cut in their x_i-part. Clearly, $\delta(r) = \delta_x(r) \cup \delta_{x_i}(r)$. We denote by $\delta_1(r)$ the set $\delta(r)$ without the most expensive edge (i.e., $\delta_1(r) = \delta(r) \setminus \{\operatorname{argmax}_{e \in \delta(r)} c(e)\}$), and for $l > 1$ we denote by $\delta_l(r)$ the set $\delta_{l-1}(r)$ without the most expensive edge (i.e., $\delta_l(r) = \delta_{l-1}(r) \setminus \{\operatorname{argmax}_{e \in \delta_{l-1}(r)} c(e)\}$). Note that for every $r \in [0, 1]$, the set $\delta_{h-1}(r)$ is an h-route cut between t_i and s. For a set $E' \subseteq E$ of edges we define $c(E') = \sum_{e \in E'} c(e)$. For a graph (resp. node set) H, let $V(H)$ be the set of nodes and $E(H)$ be the set of edges in H (resp. the set of edges in E that have both endpoints in H).

3.1 2-Route Cuts

To outline our general approach in a simple setting, we sketch in this subsection an alternative proof of the known result for 2-route single-source cuts.

In iteration i we define $\mathcal{R} = \{r \in [0, 1] \mid |\delta_{x_i}(r)| \leq 1\}$ and observe that the measure of this set is at least 1/2. For $r \in [0, 1]$, let $f(r) = \phi/k + \int_{\rho \in \mathcal{R} \cap [0,r]} c(\delta_x(\rho))\, \mathrm{d}\rho$ and $g(r) = c(\delta_1(r))$ where ϕ denotes the optimal objective value of the LP relaxation. The functions f (volume) and g (cut size) satisfy the assumptions of Lemma 1 and thus, there exists $r \in \mathcal{R}$ such that $c(\delta_1(r)) = O(\log k) f(r)$. This is the key observation of Barman and Chawla [2] (proved in a different way). We add the edges from $\delta_1(r)$ to the 2-route cut that we construct, remove the ball $B(r)$ from the graph (observe that after the removal of $\delta_1(r)$, no terminal t_j in $B(r)$ is 2-connected with s) and proceed with the next iteration. The relationship between $c(\delta_1(r))$ and $f(r)$ makes it possible to charge the cost of the edges in $\delta_1(r)$ to the volume $f(r)$ of the ball $B(r)$ (cf. the analysis of the classical 1-route cut algorithm [11]). This immediately yields the $O(\log k)$-approximation for the 2-route single-source cut problem and, with some effort, also the $O(\log^2 k)$-approximation for the general 2-route cut problem.

3.2 3-Route Cuts

In contrast to the cases $h \leq 2$, for $h = 3$ we will need to charge more than one cut to some edges. In order to keep track of how many cuts were already charged to which edge, we maintain for every edge e a counter called *a level of an edge*, denoted $\ell(e)$, which represents (an upper bound on) how many cuts were already charged to the edge e. The edges with positive level are called *restricted edges* and are maintained in a set D. Initially, the level of every edge is zero and $D = \emptyset$. Throughout the run of the algorithm, every edge $e \in D$ satisfies $x(e) \leq 1/(2h \log k)$ and $\ell(e) \leq L$, for $L = \log k$.

Recall that $F \subseteq E$ is the edge set in which we collect the edges for the final 3-route cut output by the algorithm. Whenever we have a statement holding for any h-route cut, we use the h instead of 3 so that it becomes clear which techniques only apply to $h = 3$ and which techniques could also be applied to larger h-values.

Consider the iteration of the algorithm in which we deal with the terminal t_i. For any edge $uv \in E$ let the *distance* of uv from t_i be defined as $d(uv) = \min\{d_{x+x_i}(u), d_{x+x_i}(v)\}$. We partition the edges from D into two subsets, according to their levels and their distance d from t_i:

$$D_1 = \{e \in D \mid d(e) \text{ is minimal among all } f \in D \text{ with } \ell(f) = \ell(e)\} \quad \text{and} \quad D_2 = D \setminus D_1$$

Ties are broken arbitrarily to ensure that there is at most one edge per level in D_1. Observe that for every edge $f \in D_2$ there exists an edge $e \in D_1$ with $\ell(e) = \ell(f)$.

A radius $r \in [0, 1]$ is *forbidden* if $|\delta_{x_i}(r)| > h - 1$ or if there exists an edge $e \in D_1$ such that $e \in \delta_x(r)$. A radius $r \in [0, 1]$ that is not forbidden is *good*. Let $\mathcal{R}$ denote the set of good radii for the current iteration, that is, $\mathcal{R} = \{r \in [0, 1] \mid \delta_{x_i}(r) \leq h - 1 \text{ and } \delta_x(r) \cap D_1 = \emptyset\}$.

▶ **Lemma 2.** *The measure of the set $\mathcal{R}$ of good radii is at least $1/(2h)$.*

Proof. Let μ be the measure of the set $\{r \in [0, 1] \mid |\delta_{x_i}(r)| \geq h\}$. Considering the constraint $\sum_{e \in E} x_i(e) \leq h - 1$ we obtain an upper bound on μ: $h\mu \leq \sum_{e \in E} x_i(e) \leq h - 1$, and thus, $\mu \leq 1 - 1/h$. Therefore the measure of the set $\{r \in [0, 1] \mid |\delta_{x_i}(r)| \leq h - 1\}$ is at least $1/h$. Since the number of edges in $\mathcal{D}_1$ is at most $\log k$ and since $x(e) \leq 1/(2h \log k)$ for every $e \in \mathcal{D}_1$, the measure of the set $\{r \in [0, 1] \mid \delta_x(r) \cap D_1 \neq \emptyset\}$ is at most $1/(2h)$. Hence, $|\mathcal{R}| \geq 1/(2h)$. ◀

Recall that ϕ is the optimal value of the objective function. For $r \in [0, 1]$, we define

$$V(r) = \frac{\phi}{k} + \int_{\rho \in \mathcal{R} \cap [0,r]} c(\delta_x(\rho)) d\rho \ .$$

The value $V(r)$ is called the *volume* of the ball $B(r)$. Observe that only the x-parts of the edges in the ball contribute to the volume and the x-parts of the edges from D_1 do not contribute.

Clearly, 2ϕ is an upper bound on any $V(r)$. Since $c(\delta_x(r)) \geq 0$ and $V(r) \geq \phi/k$ for all $r \in \mathcal{R}$ and $c(\delta_x(r))$ is a step function (i.e., a piece-wise constant function) on $\mathcal{R}$ with at most $2m$ jumps, where $m = |E|$, we obtain the following lemma.

▶ **Lemma 3.** *The function $V(r)$ satisfies the following properties:*

- *$V(r)$ is a nondecreasing piece-wise linear function on $\mathcal{R}$ and $V(r) > 0$ for all $r \in \mathcal{R}$,*
- *$V(r)$ is differentiable on $\mathcal{R}$, except for finitely many points,*
- *for each $r \in \mathcal{R}$, $V'(r) \geq c(\delta_{h-1}(r))$, except for finitely many points,*
- *the maximum ratio between two values of the function $V(r)$ on $\mathcal{R}$ is at most $2k$.*

Proof. Follows from the definitions of the set $\mathcal{R}$ and of the function $V(r)$. ◀

▶ **Lemma 4.** *There exists an $r \in \mathcal{R}$ such that $c(\delta_{h-1}(r)) \leq 2h\log(2k) \cdot V(r)$. Moreover, such a radius can be computed in polynomial time.*

Proof. By Lemma 2, we know that $|\mathcal{R}| \geq 1/(2h)$. Lemma 3 guarantees that we can apply Lemma 1 to the functions $f(r) = V(r)$ and $g(r) = c(\delta_{h-1}(r))$ on $\mathcal{R}$. Thus, there is an $r \in \mathcal{R}$ with $c(\delta_{h-1}(r)) \leq 2h\log(2k)V(r)$.

Since $V(r)$ is a piece-wise linear function on $\mathcal{R}$ and $c(\delta_{h-1}(r))$ is a piece-wise constant function on $\mathcal{R}$ with at most $2m$ pieces, we can efficiently find the value r for which $c(\delta_{h-1}(r))/V(r)$ is minimal, and by the first part of this lemma, this ratio is at most $2h\log(2k)$. ◀

In the current iteration, we first compute the radius r from Lemma 4 and add the edges from $\delta_2(r)$ to F (our final cut). Similar to the case of 2-route cuts, the relation between $c(\delta_2(r))$ and $V(r)$ (Lemma 4) makes it possible to charge the cost of the cut $\delta_2(r)$ to the volume $V(r)$ of the ball $B(r)$. Note that *nothing is charged to any edge* $e \in D_1$ since their x-parts do not contribute to $V(r)$. Before we proceed with the next iteration, we locally modify the graph G as described in the rest of this section.

Consider the set of edges in $\delta(r) \setminus \delta_2(r)$ and let Z be the set of endpoints of these edges that are *not* in $B(r)$. If $|Z| \leq 1$, we remove $B(r)$ and $E(B(r))$ from G and proceed with the next iteration. We can do so because for any $t_j \in B(r)$ we already constructed a 3-route cut, and for any $t_j \in V \setminus B(r)$ any path from t_j to s that goes through $B(r)$ can be reduced so that it does not contain any node from $B(r)$. If $|Z| = 2$, we define H to be a subgraph of G with vertex set $V(H) = B(r) \cup Z$ and edge set $E(H) = \{xy \in E(G) \mid x, y \in B(r)\} \cup (\delta(r) \setminus \delta_{h-1}(r))$. The two nodes in Z are called the *entry nodes* of H and the two edges in $\delta(r) \setminus \delta_{h-1}(r)$ the *entry edges* of H. Let v_i, w_i denote the two entry nodes of H, let $d_y(v_i, w_i, H)$ denote the length of the shortest path (with respect to the length function $y : E \to \mathbb{R}_{\geq 0}$) between v_i and w_i in H and let $\text{mincut}(v_i, w_i, H)$ denote the minimum cut between v_i and w_i in H. If $d_x(v_i, w_i, H) > 1/(2h\log k)$, we add the edges from $\text{mincut}(v_i, w_i, H)$ to F, charge the cost of this cut to the volume of H and remove the subgraph H from the current graph G (with the same justification as for $|Z| \leq 1$); by the *volume* of H we mean $\bar{V}(r) = \sum_{e \in E(H)} c(e)x(e)$.

▶ **Lemma 5.** *If $d_x(v_i, w_i, H) > 1/(2h\log k)$ then $c(\text{mincut}(v_i, w_i, H)) < 2h\log k \cdot \bar{V}(r)$.*

Proof. Suppose that $d_x(v_i, w_i, H) > 1/(2h\log k)$ and let $\gamma = c(\text{mincut}(v_i, w_i, H))$. Then it holds for all $\rho \in [0, d_x(v_i, w_i)]$ that $c(\delta(v_i, \rho)) > \gamma$ where $\delta(v_i, \rho)$ is the set of edges crossing distance ρ from v_i in H. Therefore,

$$\bar{V}(r) \geq \int_{\rho=0}^{d_x(v_i,w_i,H)} c(\delta(v_i, \rho)) \geq \gamma \cdot d_x(v_i, w_i) > \gamma/(2h\log k)$$

Hence, $c(\text{mincut}(v_i, w_i, H)) < 2h\log k \cdot \bar{V}(r)$. ◀

If $d_x(v_i, w_i, H) \leq 1/(2h\log k)$, we replace H in G by a new edge $v_i w_i$ and set

$$\begin{aligned} x(v_i w_i) &= d_x(v_i, w_i, H)\ , \qquad (2)\\ x_j(v_i w_i) &= d_{x+x_j}(v_i, w_i, H) - d_x(v_i, w_i, H)\ , \forall j > i,\\ c(v_i, w_i) &= c(\text{mincut}(v_i, w_i, H)),\\ \ell(v_i w_i) &= \max\{\ell_1, \ell_2\} \end{aligned}$$

where $\ell_1 = 0$ if $E(H) \cap D_1 = \emptyset$ and $\ell_1 = \max_{e \in E(H) \cap D_1} \ell(e)$ otherwise, and $\ell_2 = 0$ if $E(H) \cap D_2 = \emptyset$ and $\ell_2 = 1 + \max_{e \in E(H) \cap D_2} \ell(e)$ otherwise.

▶ **Lemma 6.** *Replacing the subgraph H by the new edge v_iw_i as described above does not increase the total volume $\sum_e x(e)c(e)$ of the system. Moreover, after the replacement all constraints of the LP are satisfied.*

Proof. The product $d_x(v_i, w_i, H) \cdot c(\text{mincut}(v_i, w_i, H))$ is a lower bound on the volume of H. The claim about the LP constraints is clear from the description of the replacement. ◀

We say that the new edge v_iw_i *represents* the edges in $E(H)$ (to be more precise, v_iw_i represents all edges that were represented by edges in $E(H)$; an edge from the original edge set represents itself). The new edge v_iw_i is added to the set D of restricted edges and all edges in D that are incident to a node in $B(r)$ are removed from D. When some cut is charged later to this new edge v_iw_i, then the charge is redistributed recursively to the edges represented by v_iw_i, proportionally to their volume. When the edge v_iw_i is cut later, then it means cutting all edges in $\text{mincut}(v_i, w_i, H)$. With this convention, every 3-route cut in the modified graph corresponds to a 3-route cut of the same cost in the original graph. We observe several things.

▶ **Lemma 7.** *For each $e \in D$, every edge represented by e was charged at most $\ell(e) + 1$ times due to the $\delta_2(r)$ cuts.*

Proof. By construction, every time something is charged to an edge f of level $\ell(f)$, either the edge is removed from the graph, or the level of the new edge that represents f is set to $\ell(f) + 1$ at least. ◀

▶ **Lemma 8.** *For each $e \in D$, $\ell(e) \leq \log k$.*

Proof. By construction, the only possibility for an increase of the maximum level of edges in D is when the level of a new edge v_iw_i is set to $\ell_2 \geq 1$, according to the definition (2). Note that in this case, at least two edges of level $\ell_2 - 1$ are removed from D (and from G). Since for every commodity i we add at most one edge to D, the claim follows. ◀

▶ **Lemma 9.** *For each $e \in D$, $x(e) \leq 1/(2h \log k)$.*

Proof. By the construction and the definition (2) of $x(v_iw_i)$. ◀

The lemmas above guarantee that the set D entering the next iteration satisfies our assumptions listed at the beginning of this section.

▶ **Theorem 10.** *The approximation ratio of the algorithm for the 3-route single-source cut problem is $O(\log^2 k)$.*

Proof. First of all, notice that if some $\text{mincut}(v_i, w_i, H)$ is charged to an edge e, then e will be removed together with H from the system and never be charged again. Hence, Lemma 5 implies that the cost of the part of F that is due to mincuts is at most $O(\phi \log k)$.

It remains to bound the cost of the h-route cuts. By the construction, the cost of every h-route cut $\delta_2(r)$ is charged to the volume $V(r)$ of some ball $B(r)$. By Lemmas 4, 6, 7 and 8, the sum of volumes of all balls to which some h-route cut was charged is at most $O(\phi \log k)$. Thus, by Lemma 4 the total cost of the h-route cuts is at most $O(\phi \log^2 k)$. ◀

4 Multiple Sources

The algorithm for multiple sources is an extension of the single-source algorithm for $h = 3$. Again, it works in iterations. In iteration i the algorithm constructs the ball B around one of the terminals s_i and t_i. In contrast to the single-source problem, there might be commodities with both terminals *inside* B. To deal with these pairs, the algorithm is recursively run in the ball B, with levels re-initialized to 0. There are two main issues that must be addressed: the number of recursive calls working with the same part of the original graph, and the (in)dependence of the subproblems. A minor change from the single-source algorithm is that now we require that $x(e)\%le1/(6h\log k)$ for every $e \in D$.

4.1 Number of overlapping recursive calls.

There are two ways how two recursive calls may work in the same area of the original graph G: (i) One of the two calls is invoked inside the other call. (ii) When the recursive call for B is completed, B is replaced by an edge and the edge is later included in a new ball B' for which another recursive call is invoked.

To guarantee that the depth of the recursion is small, we ensure that every constructed ball contains at most half of the remaining commodities. Then the depth of the recursion is $\log k$ only. In this part of our algorithm and its analysis we use the ideas from the recent paper by Barman and Chawla [2]. Lemma 11 deals with this problem.

To guarantee that there are not too many later recursive calls working in a particular area of the original graph, we apply a lazy strategy: instead of invoking the recursive call immediately after the ball B is defined, the algorithm postpones the call. If the algorithm later defines another ball B' in which the recursive call is to be run, and the ball B (the edge created from B) is contained in B', it is sufficient to perform only the recursive call for B'; this call will take care also about all commodities inside B (note that every two balls are either disjoint, or one of them is contained in the other).

Before we state and prove Lemma 11 we need a few more definitions. To simplify them, in addition to the assumption made in the previous section (i.e., $x(uv) + x_i(uv) = |d_{x+x_i}(v) - d_{x+x_i}(u)|$ for each $uv \in E$), we assume, without loss of generality, that $d_{x+x_i}(s_i) = 1$. Then, for $r \in (0,1)$ and $z \in \{s_i, t_i\}$ we define $B^z(r) = \{u \in V \mid d_{x+x_i}(z,u) \leq r\}$, $\delta^{t_i}(r) = \delta(r)$, $\delta^{s_i}(r) = \delta(1-r)$, $\delta^{t_i}_x(r) = \delta_x(r)$, and $\delta^{s_i}_{x_i}(r) = \delta_{x_i}(1-r)$ where for each $u \in V$, $d_{x+x_i}(t_i, u) = d_{x+x_i}(u)$ and $d_{x+x_i}(s_i, u) = 1 - d_{x+x_i}(u)$. For $z \in \{s_i, t_i\}$ we also define $\delta^z_1(r) = \delta^z(r) \setminus \{\operatorname{argmax}_{e \in \delta^z(r)} c(e)\}$, $\delta^z_2(r) = \delta^z_1(r) \setminus \{\operatorname{argmax}_{e \in \delta^z_1(r)} c(e)\}$ and

$$
\begin{aligned}
D^z_1 &= \{uv \in D \mid \ell(uv) = -1 \text{ or } d(z, uv) \text{ is minimal among all } e \in D \text{ with } \ell(e) = \ell(uv)\} , \\
D^z_2 &= D \setminus D^z_1 .
\end{aligned}
$$

where $d(t_i, uv) = d(uv)$ and $d(s_i, uv) = 1 - d(t_i, uv)$. Finally, for $z \in \{s_i, t_i\}$ and $r \in [0,1]$, we define $V(z,r) = \phi/k + \int_{\rho \in \mathcal{R}^z \cap [0,r]} c(\delta^z_x(\rho)) d\rho$, where $\mathcal{R}^z = \{r \in [0,1] \mid \delta_{x_i}(r) \leq h-1,\ \delta^z_x(r) \cap D^z_1 = \emptyset\}$.

▶ **Lemma 11.** *There exist good radii r_s and r_t such that $r_s + r_t \leq 1$,*

$$c(\delta_{h-1}(s_i, r_s)) \leq 3h\log(2k) \cdot V(s_i, r_s) \quad \textit{and} \quad c(\delta_{h-1}(t_i, r_t)) \leq 3h\log(2k) \cdot V(t_i, r_t) .$$

Proof. From Lemma 2 it follows that the measure of the set $\{r \in [0,1] \mid |\delta_{x_i}(r)| \leq h-1\}$ is at least $1/h$. Since the number of edges in D_1 is at most $2\log k$ and $x(e) \leq 1/(6h\log k)$ for every $e \in D_1$, the measure of the radii forbidden due to edges in D_1 is at most $1/(3h)$. As $d_{x+x_i}(t_i, s_i) = 1$, there is a radius r so that $|\mathcal{R}^{s_i} \cap [0,r]| \geq 1/(3h)$ and $|\mathcal{R}^{t_i} \cap [r,1]| \geq 1/(3h)$. It follows from Lemma 1 that there is an $r_s \in \mathcal{R}^{s_i} \cap [0,r]$ with $c(\delta^{s_i}_{h-1}(r_s)) \leq 3h\log(2k) \cdot V(s_i, r_s)$ and an $r_t \in \mathcal{R}^{t_i} \cap [r,1]$ with $c(\delta^{t_i}_{h-1}(r_t)) \leq 3h\log(2k) \cdot V(t_i, r_t)$. Since $r_s + r_t \leq 1$, the lemma follows. ◀

If r_s and r_t are the radii from Lemma 11 then the sets $B^{s_i}(r_s)$ and $B^{t_i}(r_t)$ are disjoint; thus at least one of them contains at most half of the remaining commodities. We always pick such a ball in our algorithm.

▶ **Corollary 12.** *The depth of the recursion is at most* $\log k$.

4.2 Independence of the Balls

Note that without some special care, the recursive subproblems are *not* independent as the inner part of every ball B is connected to the outside part by two edges. This is in contrast to the case $h = 2$ where the two parts of the graph are connected by a single edge and thus can be treated independently in order to deal with those commodities with both terminals in the same part of the graph. A new type of edges, *forbidden edges*, will help us to control the dependencies.

In the algorithm for the single-source multi-route cut, the input for iteration i consists not only of the current graph with the set of commodities and the corresponding fractional solution of the linear program but also of the set of restricted edges inside of this graph, which helps us to control the dependencies between the iterations. For multiple sources, besides the restricted edges, we will also use the *forbidden* edges. Formally, they will be part of the set of restricted edges but their level will be -1 and the restrictions imposed on them are stronger: they are never cut (be it an h-route cut or a mincut) and they are never charged for any cut.

Assume that we plan to invoke a recursive call for a ball built around the terminal $z \in \{s_i, t_i\}$ with radius r. We distinguish two cases (as in the previous section, v_i and w_i denote the two entry nodes of H):

- If $d_x(v_i, w_i, G \setminus H^z) \leq 1/(6h\log k)$, the recursive call is invoked for the subgraph H^z with an extra edge v_iw_i with $x(v_iw_i) = d_x(v_i, w_i, G \setminus H^z)$, $c(v_iw_i) = c(\text{mincut}(v_i, w_i, G \setminus H^z))$ and level -1. For each $j > i$ we also set $x_j(v_iw_i) = d_{x+x_j}(v_i, w_i, G \setminus H^z) - x(v_iw_i)$. The set of commodities consists of those with both terminals in $B^z(r)$, and the set of restricted edges is $(D \cap E(H^z)) \cup \{v_iw_i\}$. Since the x-length of the new edge is very short and each recursion creates at most one such edge, it is possible to impose such restrictions.
- If $d_x(v_i, w_i, G \setminus H^z) > 1/(6h\log k)$, for the recursive call we use the ball $B^z(r)$s. The set of commodities consists again of those with both terminals in $B^z(r)$, and the set of edges with restriction is $D \cap E(H^z)$. The pair $\{v_i, w_i\}$ is added to the set $\mathcal{T}$. At the very end of the algorithm, we disconnect all pairs that are in $\mathcal{T}$.

4.3 Putting it Together

Similarly to the singe-source version of the algorithm, for every part of the set F that the algorithm constructs, the ratio between the cost and the volume is bounded by $O(\log k)$, and to each part of the volume we charge at most $O(\log k)$ times within each recursive call. The only problem is that the set F that was constructed so far need not be a valid h-route cut. The difficulty is with the recursion.

In the recursive calls, when the distance $d_x(v_i, w_i, G \setminus H^z)$ was large (see the previous subsection), we ignored the fact the v_i and w_i were possibly connected *outside* the ball $B^z(r)$. Thus, at this point we have no guarantee that the set F that we constructed so far is a

3-route cut. To ensure that we do have a 3-route cut, we remove an additional set of edges from the graph. To be more specific, it suffices to disconnect the pairs of vertices in $\mathcal{T}$.

We proceed as follows. Consider the instance of the classical multicut problem consisting of the graph $G \setminus D$ and of the set $\mathcal{T}$ that represents the commodities. By construction of the set $\mathcal{T}$, the x-distance between terminals of every pair in $\mathcal{T}$ is at least $1/(6h \log k)$. Thus, if we scale the x-values by $6h \log k$, we get a fractional solution for the multicut problem for this instance. We apply the classical ball-growing rounding algorithm [6] to obtain an $O(\log k)$-approximation of the minimum multicut for this instance. Due to the scaling, the cost of the obtained cut is upper-bounded by $O(h \log^2(k) \cdot \phi)$. We add all edges from this cut to the set F.

Considering the explanation at the beginning of this section and the bound from the previous paragraph, the cost of the set F is $O(\log^3 k \cdot \phi)$, and at this point, F is a valid 3-route cut. The main theorem follows.

▶ **Theorem 13.** *The approximation ratio of the algorithm for the general* 3*-route cut problem is* $O(\log^3 k)$*.*

5 Duality of Multicommodity Multiroute Flows and Cuts

Recall that an *elementary* h*-flow* between s and t is a set of h edge-disjoint paths between s and t, each carrying a unit flow. Let $\mathcal{Q}_i$ denote the set of all elementary h-flows between s_i and t_i and let $\mathcal{Q} = \bigcup_{i=i}^{k} \mathcal{Q}_i$. Then the problem of finding a maximum multicommodity h-route flow has the following linear programming formulation; there is a non-negative variable $f(q)$ for every $q \in \mathcal{Q}$ where the value $f(q)$ represents the total amount of flow sent along the h-route flow q. On the right side of the page we state the dual linear program.

$$\begin{aligned} \max \sum_{q \in \mathcal{Q}} f(q) & & & \qquad (3) \\ \sum_{q \in \mathcal{Q}: e \in q} f(q) &\leq h \cdot c(e) & \forall e \in E & \\ f(q) &\geq 0 & \forall q \in \mathcal{Q} & \end{aligned}$$

$$\begin{aligned} \min\ h \cdot \sum_{e \in E} c(e) \cdot x(e) & & & \qquad (4) \\ \sum_{e \in q} x(e) &\geq 1 & \forall q \in \mathcal{Q} & \\ x(e) &\geq 0 & \forall e \in E & \end{aligned}$$

Note that without the factor h in the objective function the linear program (4) is another relaxation of the h-route cut problem (the approximation algorithm of Chekuri and Khanna [4] for 2-route cuts is based on this relaxation). We will refer by (4') to the linear program (4) with the objective function scaled down to $\sum_{e \in E} c(e) \cdot x(e)$.

There are simple examples showing that the linear relaxation (4') is by a factor of h lower (asymptotically) than the linear relaxation of (1). Think about two vertices s and t connected by M parallel edges. Then the fractional optimum for the linear program (4') is M/h (assign a value $1/h$ to every variable) while the fractional optimum of the linear program (1) is $M - h$.

The main technical result of this section is that the gap between the two relaxations is not more than h. A corollary of this result is an approximate duality theorem for multiroute cuts and flows.

▶ **Theorem 14.** *Given an instance of the* h*-route cut problem, let* O_1 *denote the optimum value of the linear program (1) and* O_2 *the optimum value of the linear program (4'). Then* $O_2 \leq O_1 \leq h \cdot O_2$*, and the bound is tight.*

Proof. Since the first inequality is trivial, it suffices to prove the second one. Let x be an optimum solution of the linear program (4'). We are going to derive from x a solution $\bar{x}, x_1, \ldots, x_k \in \mathbb{R}^E$ of the linear program (1) with the objective value being larger by a factor of at most h (i.e., $\sum_{e\in E} c(e)\bar{x}(e) \le h \sum_{e\in E} c(e)x(e)$). For each $e \in E$, let $\bar{x}(e) = h \cdot x(e)$. It suffices to prove that for each i, the following linear program has a feasible solution x_i. As in Section 2, $\mathcal{P}_i$ denotes the set of all paths between s_i and t_i.

$$\begin{aligned} \sum_{e\in p} x_i(e) &\ \ge\ 1 - \sum_{e\in p} \bar{x}(e) \quad \forall p \in \mathcal{P}_i \\ \sum_{e\in E} x_i(e) &\ \le\ h-1 \qquad \forall e \in E \\ x_i(e) &\ \ge\ 0 \qquad \forall e \in E \end{aligned} \tag{5}$$

Assume, for a contradiction, that the linear program (5) does not have a feasible solution. Then, by Farkas' lemma, there exists a non-negative vector $\lambda \in \mathbb{R}^{\mathcal{P}_i}$ and a non-negative scalar γ such that

$$\begin{aligned} \sum_{p\in\mathcal{P}_i : e\in p} \lambda(p) &\ \le\ 1 \quad \forall e \in E \\ \sum_{p\in\mathcal{P}_i} \lambda(p)(1 - \sum_{e\in p} \bar{x}(e)) &\ >\ h-1 \end{aligned} \tag{6}$$

(without loss of generality, we assume that $\gamma = 1$; note that every vector (λ, γ) obtained by the application of the Farkas' to the linear program (5) satisfies $\gamma > 0$ and thus, we can scale the (λ, γ) to guarantee $\gamma = 1$). In the following discussion, among all vectors λ satisfying the constraints (6) we fix the one for which $\sum_{p\in\mathcal{P}_i} \lambda(p)$ is minimal.

Observe that λ corresponds to a feasible flow between s_i and t_i in the graph G with all edge capacities set to one; the size of the flow is at least $h - 1 + \sum_{p\in\mathcal{P}_i} \sum_{e\in p} \lambda(p)\bar{x}(e) > h-1$. For each edge $e \in E$, let $\lambda(e) = \sum_{p:e\in p} \lambda(p)$ and let $E' = \{e \in E \mid \lambda(e) > 0\}$ be the subset of edges on which the flow λ is non-zero. Since the flow is realized in a graph with unit capacities and the size of the flow is strictly larger than $h-1$, by Mengers' theorem there exist h edge disjoint paths between s_i and t_i in (V, E'); let $q \in \mathcal{Q}_i$ denote the corresponding elementary h-flow and $\lambda(q) = \min_{e\in q} \lambda(e)$. Let $\lambda' \in \mathbb{R}^{\mathcal{P}_i}$ be (a path-decomposition of) the flow obtained from the flow λ by subtracting $\lambda(q)$ units of flow from every edge $e \in q$. Note that $\sum_{p\in\mathcal{P}_i} \lambda(p) > \sum_{p\in\mathcal{P}_i} \lambda'(p)$. Since we started with a feasible solution x of the linear program (4'), from the definition of $\bar{x}$ we know that $\sum_{e\in q} \bar{x}(e) \ge h$. Observing that

$$\sum_{p\in\mathcal{P}_i} \lambda(p)(1 - \sum_{e\in p} \bar{x}(e)) = \sum_{p\in\mathcal{P}_i} \lambda'(p)(1 - \sum_{e\in p} \bar{x}(e)) + \lambda(q)(h - \sum_{e\in q} \bar{x}(e)) \ ,$$

we conclude that $\sum_{p\in\mathcal{P}_i} \lambda'(p)(1 - \sum_{e\in p} \bar{x}(e)) > h-1$. However, this is a contradiction with the choice of λ: the flow λ' also satisfies the constraints (6) and its size is smaller than the size of λ. Thus, the linear program (5) has a feasible solution, for each i, and the proof is completed. ◀

▶ **Corollary 15** (Duality of multiroute multicommodity flows and cuts). *For any instance with k commodities, the cost of the minimum h-route cut for $h \le 3$ is at least a fraction $1/h$ of the maximum h-route multicommodity flow, and is always at most $O(h^2 \log^3 k)$ times as much.*

Proof. The first relation is trivial: one always has to block at least one of the h paths of every elementary h-flow. The other relation follows from Theorem 14, the duality of the linear programs (3) and (4), and Theorem 13 (the approximation algorithm). ◀

5.1 Sparsest multiroute cut

The *sparsest multiroute cut problem* is a multiroute analog of the sparsest cut problem. By a combination of standard [7, 11] and our techniques we obtain the following results.

▶ **Theorem 16.** *The approximation ratio achievable in polynomial time for the multiroute sparsest cut problem with $h \leq 3$ is $O(h^2 \log h \log^3 k \log D)$ where $D = \sum_{i=1}^{k} d_i$.*

▶ **Corollary 17.** *For any instance with k commodities, the sparsest h-route cut for $h \leq 3$ is at least as large as the maximum concurrent h-route multicommodity flow, and is always at most $O(h^2 \log h \log^3 k\ \log D)$ times larger.*

Acknowledgments.

The first author would like to thank Jiří Sgall and Thomas Erlebach for stimulating discussions.

References

1 Amitabha Bagchi, Amitabh Chaudhary, Petr Kolman, and Jiří Sgall. A simple combinatorial proof for the duality of multiroute flows and cuts. Technical Report 2004-662, Charles University, Prague, 2004.

2 Siddharth Barman and Shuchi Chawla. Region growing for multi-route cuts. In *Proceedings of the 21th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

3 Henning Bruhn, Jakub Černý, Alexander Hall, Petr Kolman, and Jiří Sgall. Single source multiroute flows and cuts on uniform capacity networks. *Theory of Computing*, 4(1):1–20, 2008. Preliminary version in Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007.

4 Chandra Chekuri and Sanjeev Khanna. Algorithms for 2-route cut problems. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 5125 of *Lecture Notes in Computer Science*, pages 472–484, 2008.

5 L. R. Ford and D. R. Fulkerson. Maximum flow through a network. *Canad. J. Math.*, 8:399–404, 1956.

6 Naveen Garg, Vijay V. Vazirani, and Mihalis Yannakakis. Approximate max-flow min-cut theorems and their applications. *SIAM Journal on Computing*, 25(2):235–251, 1996.

7 Nabil Kahale. On reducing the cut ratio to the multicut problem. Technical report, 1993. DIMACS Technical report 93-78.

8 Wataru Kishimoto and M. Takeuchi. On m-route flows in a network. *IEICE Transactions*, J-76-A(8):1185–1200, 1993. (in Japanese).

9 Petr Kolman and Christian Scheideler. Approximate duality of multicommodity multiroute flows and cuts on uniform capacity networks. Technical report, 2010. Submitted.

10 Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *Journal of the ACM*, 46(6):787–832, November 1999. Preliminary version in Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science (FOCS), 1988.

11 David B. Shmoys. Cut problems and their application to divide-and-conquer. In Dorith S. Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*, pages 192–235. PWS Publishing Company, 1997.

Compact Visibility Representation of Plane Graphs*

Jiun-Jie Wang[1] and Xin He[2]

1 **Department of Comp. Sci. and Eng., Univ. at Buffalo, Buffalo, NY 14260-2000 USA.** jiunjiew@buffalo.edu

2 **Department of Comp. Sci. and Eng., Univ. at Buffalo, Buffalo, NY 14260-2000 USA.** xinhe@buffalo.edu

Abstract

The visibility representation (VR for short) is a classical representation of plane graphs. It has various applications and has been extensively studied. A main focus of the study is to minimize the size of the VR. It is known that there exists a plane graph G with n vertices where any VR of G requires a grid of size at least $\frac{2}{3}n \times (\frac{4}{3}n - 3)$ (width × height). For upper bounds, it is known that every plane graph has a VR with grid size at most $\frac{2}{3}n \times (2n - 5)$, and a VR with grid size at most $(n - 1) \times \frac{4}{3}n$. It has been an open problem to find a VR with both height and width simultaneously bounded away from the trivial upper bounds (namely with size at most $c_h n \times c_w n$ with $c_h < 1$ and $c_w < 2$).

In this paper, we provide the first VR construction with this property. We prove that every plane graph of n vertices has a VR with height $\leq \max\{\frac{23}{24}n + 2\lceil\sqrt{n}\rceil + 4, \frac{11}{12}n + 13\}$ and width $\leq \frac{23}{12}n$. The area (height×width) of our VR is larger than the area of some of previous results. However, bounding one dimension of the VR only requires finding a good st-orientation *or* a good dual s^*t^*-orientation of G. On the other hand, to bound both dimensions of VR simultaneously, one must find a good st-orientation *and* a good dual s^*t^*-orientation at the same time, and thus is far more challenging. Since st-orientation is a very useful concept in other applications, this result may be of independent interests.

1998 ACM Subject Classification G.2.2 Graph algorithms.

Keywords and phrases plane graph, plane triangulation, visibility representation, st-orientation

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.141

1 Introduction

Drawing plane graphs has emerged as a fast growing research area in recent years (see [1] for a survey). A *visibility representation* (VR for short) is a classical drawing style of plane graphs, where the vertices of a graph G are represented by non-overlapping horizontal line segments (called *vertex segment*), and each edge of G is represented by a vertical line segment touching the vertex segments of its end vertices. Fig. 1 shows a VR of a plane graph G. The problem of computing a compact VR is important not only in algorithmic graph theory, but also in practical applications. A simple linear time VR algorithm was given in [13, 14] for 2-connected plane graphs. It uses an *st-orientation* of G and the corresponding st-orientation of its st-dual G^* to construct a VR. Using this approach, the height of the VR is bounded by $(n - 1)$ and the width of the VR is bounded by $(2n - 5)$ [13, 14].

* Research supported in part by NSF Grant CCR-0635104.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 141–152
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

	Plane Graph		4-Connected Plane Graph	
	Width	Height	Width	Height
1	$\leq (2n-5)$ [13, 14]	$\leq (n-1)$ [13, 14]		
2	$\leq \lfloor \frac{3n-6}{2} \rfloor$ [6]			
3	$\leq \lfloor \frac{22n-42}{15} \rfloor$ [9]		$\leq (n-1)$ [7]	
4		$\leq \lfloor \frac{5n}{6} \rfloor$ [16]		
5	$\leq \lfloor \frac{13n-24}{9} \rfloor$ [17]			$\leq \lceil \frac{3n}{4} \rceil$ [15]
6		$\leq \lfloor \frac{4n-1}{5} \rfloor$ [18]		
7		$\leq \frac{2n}{3} + \lfloor 2\sqrt{n} \rfloor$ [4]		
8		$\leq \frac{2n}{3} + 14$ [19]		
9	$\leq \lfloor \frac{4n}{3} \rfloor - 2$ [3]			$\leq \lceil \frac{n}{2} \rceil + 2\lceil \sqrt{\frac{n-2}{2}} \rceil$ [2]
10			$\leq \frac{3}{2}n$ [5]	$\leq \frac{3}{4}n + 2\lceil \sqrt{n} \rceil + 4$ [5]
11	$\leq \frac{23}{12}n$	$\leq \max\{\frac{23}{24}n + 2\lceil \sqrt{n} \rceil + 4,$ $\frac{11}{12}n + 13\}$		

Table 1 Previous and new results on the height and the width of VR. (For the line 8, the original bound given in [19] was Height $\leq 2n/3 + O(1)$. By a more careful calculation, the term $O(1)$ is actually 14.)

As in many other graph drawing problems, one of the main concerns in the VR research is to minimize the grid size (i.e. the height and the width) of the representation. For the lower bounds, it was shown in [16] that there exists a plane graph G with n vertices where any VR of G requires a grid of size at least $(\lfloor \frac{2n}{3} \rfloor) \times (\lfloor \frac{4n}{3} \rfloor - 3)$. Some work has been done to reduce the height and width of the VR by carefully constructing special st-orientations. Table 1 compares related previous results and new result in this paper.

The line 1 in Table 1 gives the trivial upper bounds. All other results, except the line 10 and 11 (the recult in this paper), concentrated on one dimension of the VR (either the width or the height). In Table 1, the un-mentioned dimension is bounded by the trivial upper bound (namely, $n-1$ for the height and $2n-5$ for the width). In [11, 12], heuristic algorithms were developed aiming at reducing the height and the width of VRs simultaneously. The line 10 in Table 1 is the only VR construction with simultaneously reduced height and width. However, it only works for 4-connected plane graphs. The line 11 shows the new result in this paper: we prove that every plane graph with n vertices has a VR with height at most $\max\{\frac{23}{24}n + 2\lceil \sqrt{n} \rceil + 4, \frac{11}{12}n + 13\}$ and width at most $\frac{23}{12}n$. The representation can be constructed in linear time.

The present paper is organized as follows. Section 2 introduces preliminaries. Section 3 presents a decomposition lemma for plane graphs. Section 4 presents the construction of VR with the stated height and width. Section 5 concludes the paper.

2 Preliminaries

In this paper, we only consider simple graphs (namely without self-loops and multiple edges). A *planar graph* is a graph $G = (V, E)$ such that the vertices of G can be drawn in the plane and the edges of G can be drawn as non-intersecting curves. Such a drawing is called an *embedding*. The embedding divides the plane into a number of connected regions. Each region is called a *face*. The unbounded face is the *exterior face*. The other faces are *interior faces*. The vertices and edges that are not on the boundary of the exterior face are called interior vertices and edges, respectively. A *plane graph* is a planar graph with a fixed embedding. A *plane triangulation* is a plane graph where every face is a triangle (including the exterior face). $|G|$ denotes the number of vertices of G. $I(G)$ denotes the set of interior vertices of G. Thus $|I(G)| = |G| - 3$ for a plane triangulation G.

For a path P, length(P) (or $|P|$) denotes the number of edges in P. For two vertices a, b in P, $P(a, b)$ denotes the sub-path of P from a to b (inclusive). (We slightly abuse the notation here: For a graph G, $|G|$ denotes the number of vertices in G. For a path P, $|P|$ denotes the number of edges in P.)

When discussing VRs, we assume the input graph G is a plane triangulation. (If not, we get a triangulation G' by adding dummy edges into G. After constructing a VR for G', we can get a VR of G by deleting the vertical line segments for the dummy edges). From now on, G always denotes a plane triangulation.

A *numbering* $\mathcal{O}$ of a set $S = \{a_1, \ldots, a_k\}$ is a one-to-one mapping between S and the set $\{1, 2, \ldots, k\}$. We write $\mathcal{O} = \langle a_{i_1}, a_{i_2}, \ldots, a_{i_k} \rangle$ to indicate $\mathcal{O}(a_{i_1}) = 1$, $\mathcal{O}(a_{i_2}) = 2$ ... etc. A set S with a numbering written this way is called an *ordered list*. For two elements a_i and a_j, if a_i is assigned a smaller number than a_j in $\mathcal{O}$, we write $a_i \prec_{\mathcal{O}} a_j$. Let S_1 and S_2 be two disjoint sets. If $\mathcal{O}_1$ is a numbering of S_1 and $\mathcal{O}_2$ is a numbering of S_2, their concatenation, written as $\mathcal{O} = \langle \mathcal{O}_1, \mathcal{O}_2 \rangle$, is the numbering of $S_1 \cup S_2$ where $\mathcal{O}(x) = \mathcal{O}_1(x)$ for all $x \in S_1$ and $\mathcal{O}(y) = \mathcal{O}_2(y) + |S_1|$ for all $y \in S_2$.

G is called an *directed graph* (digraph) if each edge of G is assigned a direction. An *orientation* of a (undirected) graph G is a digraph obtained from G by assigning a direction to each edge of G. We use G to denote both the resulting digraph and the underlying undirected graph unless otherwise specified. (Its meaning will be clear from the context.)

Let $G = (V, E)$ be an undirected graph. A numbering $\mathcal{O}$ of V induces an orientation of G as follows: each edge of G is directed from its lower numbered end vertex to its higher numbered end vertex. The resulting digraph, denoted by $G_{\mathcal{O}}$, is called the *orientation derived from* $\mathcal{O}$ which, obviously, is an acyclic digraph. We use length$_G(\mathcal{O})$ (or simply length$(\mathcal{O})$ if G is clear from the context) to denote the length of the longest directed path in $G_{\mathcal{O}}$.

For a 2-connected plane graph G and an exterior edge (s, t), an orientation of G is called an *st-orientation* if the resulting digraph is acyclic with s as the only source and t as the only sink. Such a digraph is also called an *st-graph*. Lempel et al. [8] showed that for every 2-connected plane graph G and an exterior edge (s, t), there exists an st-orientation. For more properties of st-orientation and st-graph, we refer readers to [10].

Let G be a 2-connected plane graph and (s, t) an exterior edge. An *st-numbering* of G is a one-to-one mapping $\xi : V \to \{1, 2, \ldots, n\}$, such that $\xi(s) = 1$, $\xi(t) = n$, and each vertex $v \neq s, t$ has two neighbors u, w with $\xi(u) < \xi(v) < \xi(w)$, where u (w, resp.) is called a *smaller neighbor* (*bigger neighbor*, resp.) of v. Given an st-numbering ξ of G, the orientation of G derived from ξ is obviously an st-orientation of G. On the other hand, if $G = (V, E)$ has an st-orientation $\mathcal{O}$, we can define an one-to-one mapping $\xi : V \to \{1, \ldots, n\}$ by topological sort. It is easy to see that ξ is an st-numbering and the orientation derived from ξ is $\mathcal{O}$. From now on, we will interchangeably use the term "an st-numbering" of G and the term "an st-orientation" of G, where each edge of G is directed accordingly.

▶ **Definition 1.** Let G be a plane graph with an st-orientation $\mathcal{O}$, where (s, t) is an exterior edge drawn at the left on the exterior face of G. The *st-dual* graph G^* of G and the dual orientation $\mathcal{O}^*$ of $\mathcal{O}$ is defined as follows:

- Each face f of G corresponds to a node f^* of G^*. In particular, the unique interior face adjacent to the edge (s, t) corresponds to a node s^* in G^*, the exterior face corresponds to a node t^* in G^*.

- For each edge $e \neq (s,t)$ of G separating a face f_1 on its left and a face f_2 on its right, there is a dual edge e^* in G^* from f_1^* to f_2^*.
- The dual edge of the exterior edge (s,t) is directed from s^* to t^*.

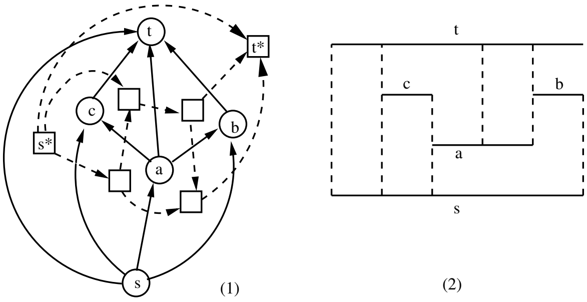


Figure 1 (1) An st-graph G and its st-dual graph G^*; (2) A VR of G.

Fig. 1 (1) shows an st-graph G and its st-dual graph G^*. (Circles and solid lines denote the vertices and the edges of G. Squares and dashed lines denote the nodes and the edges of G^*.) It is well known that the st-dual graph G^* defined above is an st-graph with source s^* and sink t^*. The correspondence between an st-orientation $\mathcal{O}$ of G and the dual st-orientation $\mathcal{O}^*$ is a one-to-one correspondence. The following theorem was given in [13, 14]:

▶ **Theorem 2.** *Let G be a 2-connected plane graph with an st-orientation $\mathcal{O}$. Let $\mathcal{O}^*$ be the dual st-orientation of the st-dual graph G^*. A VR of G can be obtained from $\mathcal{O}$ in linear time. The height of the VR is* length($\mathcal{O}$)*. The width of the VR is* length($\mathcal{O}^*$)*. Since G has n vertices and G^* has $2n-4$ nodes, any st-orientation of G leads to a VR with height $\leq n-1$ and width $\leq 2n-5$.*

Fig. 1 (2) shows a VR of the graph G shown in Fig. 1 (1). The width of the VR is length($\mathcal{O}^*$) $= 5$. The height of the VR is length($\mathcal{O}$) $= 3$.

The following theorems were given in [19, 3, 5], and will be needed later for our VR construction.

▶ **Theorem 3.** *[19] Every plane triangulation with n vertices has a VR with width $\leq 2n-5$ and height $\leq \frac{2}{3}n+14$, which can be constructed in linear time.*

▶ **Theorem 4.** *[3] Every plane triangulation with n vertices has a VR with height $\leq n-1$ and width $\leq \lfloor\frac{4}{3}n\rfloor - 2$, which can be constructed in linear time.*

▶ **Theorem 5.** *[5] Every 4-connected plane triangulation with n vertices has a VR with height $\leq \frac{3}{4}n + 2\lceil\sqrt{n}\rceil + 4$ and width $\leq \frac{3}{2}n$, which can be constructed in linear time.*

From Theorem 2, results in above theorems can also be stated in terms of the length of the orientations of G. The statement "G has an st-orientation $\mathcal{O}$ such that length($\mathcal{O}$) $\leq x$ and length($\mathcal{O}^*$) $\leq y$" is equivalent to the statement "the VR of G derived from $\mathcal{O}$ has height $\leq x$ and width $\leq y$". We will use these two statements interchangeably.

3 A Decomposition Lemma

The basic idea of our VR construction is as follows: We use the VR constructions in Theorems 2, 3, 4 and 5 for different subgraphs of G, some of them have small width and others have small height. The crux of the construction is to find a proper balance that reduces overall height and width of the VR. In this section, we prove a decomposition lemma that is needed by our VR construction to achieve the balance.

Let $G = (V, E)$ be a plane graph. A *triangle* of G is a set of three mutually adjacent vertices. The notation $\triangle = (a, b, c)$ denotes a triangle consisting of vertices a, b, c. A triangle $\triangle$ divides the plane into its interior and exterior regions. We say $\triangle = (a, b, c)$ is a *separating triangle* if $G - \{a, b, c\}$ is disconnected. In other words, $\triangle = (a, b, c)$ is a separating triangle if there are vertices in both its interior and exterior regions. The following fact by Whitney is well known:

▶ **Fact 1.** A plane triangulation G is 4-connected if and only if G has no separating triangles.

Let $\triangle = (a, b, c)$ be a separating triangle. $G_\triangle$ denotes the subgraph of G induced by $\{a, b, c\} \cup \{v \in V \mid v \text{ is in interior of } \triangle\}$. $\triangle$ is *maximal* if there is no other separating triangle $\triangle'$ such $G_\triangle \subset G_{\triangle'}$. Two triangles $\triangle_1$ and $\triangle_2$ are *related* if either $G_{\triangle_1} \subseteq G_{\triangle_2}$ or $G_{\triangle_2} \subseteq G_{\triangle_1}$.

Let G_1 and G_2 be two plane triangulations. If G_1 has an internal face f such that the vertex set of f and the vertex set of the outer face of G_2 are identical, we can *embed* G_2 *into* G_1 by identifying the face f and the exterior face of G_2. The resulting plane triangulation is denoted by $G_1 \oplus_f G_2$ (or simply $G_1 \oplus G_2$).

▶ **Definition 6.** Let G_1 and G_2 be two plane triangulations such that G_2 can be embedded into G_1 by a common face $f = \{a, b, c\}$. Let $\mathcal{O}_1$ be an st-orientation of G_1 and $\mathcal{O}_2$ be an st-orientation of G_2 such that the three edges $\{(a, b), (b, c), (c, a)\}$ are oriented the same way in $\mathcal{O}_1$ and $\mathcal{O}_2$. $\mathcal{O}_{G_1} \oplus \mathcal{O}_{G_2}$ denotes the union of $\mathcal{O}_1$ and $\mathcal{O}_2$, which is an orientation of $G_1 \oplus G_2$.

▶ **Lemma 7.** *Let G_1, G_2, $\mathcal{O}_1$, and $\mathcal{O}_2$ be as in Definition 6. Then $\mathcal{O}_{G_1} \oplus \mathcal{O}_{G_2}$ is an st-orientation of $G_1 \oplus G_2$.*

Proof. Immediate from the definition. ◀

▶ **Definition 8.** The *4-block tree* of a plane triangulation G is a rooted tree T defined as follows:

- If G has no separating triangles (i.e. G is 4-connected), then T consists of a single root r.
- If not, let $\triangle_1, \ldots, \triangle_p$ be the maximal separating triangles of G. Let T_i be the 4-block tree of $G_{\triangle_i}$. Then T is the tree with root r and the roots of T_i $(1 \leq i \leq p)$ as the children of r.

From the definition, we have the following properties:

- Each non-root node u of T corresponds to a separating triangle $\triangle_u$ of G.
- For any $u, v \in T$, u and v have ancestor-descendant relation if and only if $\triangle_u$ and $\triangle_v$ are related in G.

For a node u of T, G_u denotes the subgraph $G_{\triangle_u} - (\cup_{v \in C(u)} I(G_{\triangle_v}))$ where $C(u)$ is the set of children of u in T. In other words, G_u is obtained from $G_{\triangle_u}$ by deleting all vertices that are in the interior of the maximal separating triangles of $G_{\triangle_u}$. Since G_u has no separating triangles, G_u is 4-connected. Each G_u is called a *4-block component* of G. Fig. 2 shows a plane triangulation G, the 4-block components and the 4-block tree of G.

For a node $u \in T$, define $|T_u| = |G_{\triangle_u}|$.

▶ **Lemma 9.** *Let G be a triangulation and T be its 4-block tree. At least one of the following two conditions holds.*

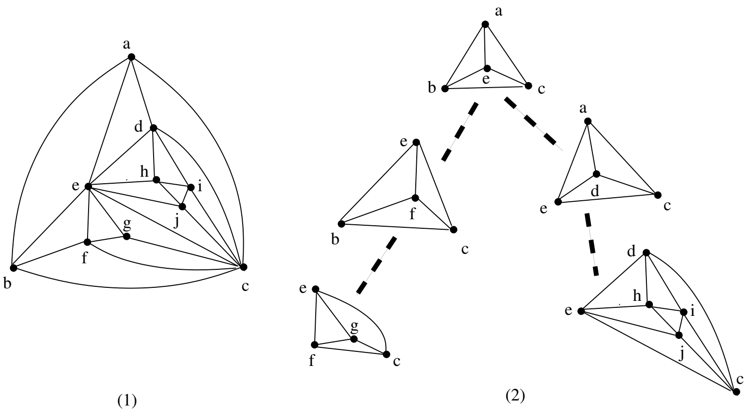


Figure 2 (1) A triangulation G; (2) 4-block components and the 4-block tree T of G.

1. *There exists a node v in T such that $|G_v| \geq \frac{n}{6}$.*
2. *There exists a set of unrelated separating triangles $\{\triangle_1, \triangle_2, \ldots, \triangle_h\}$, such that $|G_{\triangle_i}| \geq 5$ and $\frac{n}{4} - 3 \leq \sum_{i=1}^{h} |I(G_{\triangle_i})| \leq \frac{3}{4}n - 3$.*

Proof. Let r be the root of T. Let H be a maximal path in T from r to some node v of T such that for each node $u \in H$, $|T_u| \geq \frac{3n}{4}$ (v can be the root r).

If v is a leaf of T, then $|G_v| \geq \frac{3n}{4} > \frac{n}{6}$. So condition (1) is satisfied.

Now, suppose v is not a leaf. Let $\{v_1, v_2, \ldots, v_p\}$ be the children of v in T. Without loss of generality, assume $|T_{v_1}| \leq |T_{v_2}| \leq \ldots \leq |T_{v_p}|$. Then, either $\frac{n}{4} \leq |T_{v_p}| < \frac{3n}{4}$; or $|T_{v_i}| < \frac{n}{4}$ for all $v_i \in \{v_1, v_2, \ldots, v_p\}$.

If $\frac{n}{4} \leq |T_{v_p}| < \frac{3n}{4}$, then the separating triangle $\triangle_{v_p}$ satisfies $\frac{n}{4} - 3 \leq |I(G_{\triangle_{v_p}})| \leq \frac{3n}{4} - 3$. So the single separating triangle $\triangle_{v_p}$ satisfies condition (2).

Now suppose $|T_{v_i}| < \frac{n}{4}$ for all v_i. Let i_m be the index such that $|T_{v_i}| \leq 4$ for all $i \leq i_m$ and $|T_{v_i}| \geq 5$ for all $i > i_m$. There are three cases.

(1) $\sum_{i>i_m}(|T_{v_i}| - 3) < \frac{n}{4} - 3$.

Let $n_1 = |G_v|$. Since G_v is a triangulation with n_1 vertices, G_v has $2n_1 - 5$ internal faces by Euler's formula. Each child v_i of v corresponds to a maximal separating triangle of $G_{\triangle_v}$, and each such separating triangle is one of the interior faces of G_v. Thus, $i_m \leq p \leq 2n_1 - 5$. Since $|I(G_{\triangle_{v_i}})| = 1$ for all $i \leq i_m$, we have:

$$\begin{aligned} \frac{3}{4}n &\leq |T_v| = n_1 + \sum_{i \leq i_m} |I(G_{\triangle_{v_i}})| + \sum_{i>i_m} |I(G_{\triangle_{v_i}})| \\ &= n_1 + i_m + \sum_{i>i_m} |I(G_{\triangle_{v_i}})| \leq n_1 + (2n_1 - 5) + \sum_{i>i_m} |I(G_{\triangle_{v_i}})| \end{aligned}$$

From the assumption $\sum_{i>i_m} |I(G_{\triangle_{v_i}})| < \frac{n}{4} - 3$, we have: $3n_1 - 5 > \frac{3}{4}n - \frac{n}{4} = \frac{n}{2}$. This implies $|G_v| = n_1 \geq \frac{n}{6} + \frac{5}{3}$. So G_v satisfies (1).

(2) $\frac{n}{4} - 3 \leq \sum_{i>i_m}(|T_{v_i}| - 3) \leq \frac{3}{4}n - 3$.

This is equivalent to $\frac{n}{4} - 3 \leq \sum_{i>i_m} |I(G_{\triangle_{v_i}})| \leq \frac{3}{4}n - 3$. So the set of unrelated separating triangles $\{\triangle_{v_{i_m+1}}, \triangle_{v_{i_m+2}}, \ldots, \triangle_{v_{i_p}}\}$ satisfies (2).

(3) $\sum_{i>i_m}(|T_{v_i}| - 3) > \frac{3}{4}n - 3$

Let i_t be the first index such that $\sum_{i_m<i\le i_t}(|T_{v_i}|-3) \ge \frac{n}{4}-3$. Because each $|T_{v_i}| < \frac{n}{4}$, clearly $\sum_{i_m<i\le i_t}(|T_{v_i}|-3) \le \frac{3}{4}n-3$. So the set of unrelated separating triangles $\{\triangle_{v_{i_m+1}}, \triangle_{v_{i_m+2}}, \ldots, \triangle_{v_{i_t}}\}$ satisfies (2). ◀

4 Compact Visibility Representation

In this section. we describe our compact VR construction of a plane triangulation G. In order to reduce VR's height and width simultaneously, we construct a VR of G by using different VRs for some subgraphs of G. As stated in theorems 2, 3, 4 and 5, some of these VRs have small height and others have small width. Roughly speaking, we select a set of separating triangles, $\{\triangle_1, \triangle_2, \ldots, \triangle_h\}$ of G. For the subgraph of G that is outside of $\{G_{\triangle_1}, G_{\triangle_2}, \ldots, G_{\triangle_h}\}$ (call it G'), we use a VR of G' with small height. For each $G_{\triangle_i}$, we use a VR with small width. Then, we embed each $G_{\triangle_i}$ into G'.

Define $\mathcal{X}(k) = \lceil \frac{2}{3}k \rceil - 1$. It is easy to verify:

- $\mathcal{X}(k)$ is a non-decreasing function; and $\mathcal{X}(k) \ge 1$ and $\mathcal{X}(k) \ge k/3$ for all $k \ge 2$.

▶ **Theorem 10.** *Let $S = \{\triangle_1, \triangle_2, \ldots, \triangle_h\}$ be a set of unrelated separating triangles of G. Then G has an st-orientation $\mathcal{O}$ such that* $\text{length}(\mathcal{O}) \le \frac{2n}{3} + \frac{\sum_{i=1}^{h} |I(G_{\triangle_i})|}{3} + 14$ *and* $\text{length}(\mathcal{O}^*) \le 2n - 5 - \sum_{i=1}^{h} \mathcal{X}(|I(G_{\triangle_i})|)$.

Proof. Define $G_{ext} = G - \cup_{i=1}^{h} I(G_{\triangle_i})$ and $G_j = G_{ext} \cup (\cup_{i=1}^{j} G_{\triangle_i})$. We will show that G_j $(0 \le j \le h)$ has an st-orientation $\mathcal{O}_j$ so that:

▶ **Claim 1.** $\text{length}(\mathcal{O}_j) \le \frac{2}{3}|G_j| + \frac{\sum_{i=1}^{j} |I(G_{\triangle_i})|}{3} + 14$.

▶ **Claim 2.** $\text{length}(\mathcal{O}_j^*) \le 2|G_j| - 5 - \sum_{i=1}^{j} \mathcal{X}(|I(G_{\triangle_i})|)$.

Then the theorem follows. We prove claims 1 and 2 by induction.

Base case $j = 0$: From Theorem 3, $G_0 = G_{ext}$ has an st-orientation $\mathcal{O}_0$ such that $\text{length}(\mathcal{O}_0) \le \frac{2}{3}|G_0| + 14$ and $\text{length}(\mathcal{O}_0^*) \le 2|G_0| - 5$. So the claims hold for the base case.

Induction hypothesis: G_k has an st-orientation $\mathcal{O}_k$ such that: $\text{length}(\mathcal{O}_k) \le \frac{2}{3}|G_k| + \frac{\sum_{i=1}^{k} |I(G_{\triangle_i})|}{3} + 14$. and $\text{length}(\mathcal{O}_k^*) \le 2|G_k| - 5 - \sum_{i=1}^{k} \mathcal{X}(|I(G_{\triangle_i})|)$.

Suppose that $\triangle_{k+1} = \{a_{k+1}, b_{k+1}, c_{k+1}\}$. Without loss of generality, assume the edges of $\triangle_{k+1}$ are oriented in $\mathcal{O}_k$ as $\{(a_{k+1} \to b_{k+1}), (b_{k+1} \to c_{k+1}), (a_{k+1} \to c_{k+1})\}$.

By Theorem 4, $G_{\triangle_{k+1}}$ has an st-orientation $\mathcal{O}_{\triangle_{k+1}}$, with a_{k+1} as the source and c_{k+1} as the sink, such that: $\text{length}(\mathcal{O}_{\triangle_{k+1}}) \le |G_{\triangle_{k+1}}| - 1$ and $\text{length}(\mathcal{O}^*_{\triangle_{k+1}}) \le \lfloor \frac{4}{3}|G_{\triangle_{k+1}}| \rfloor - 2$.

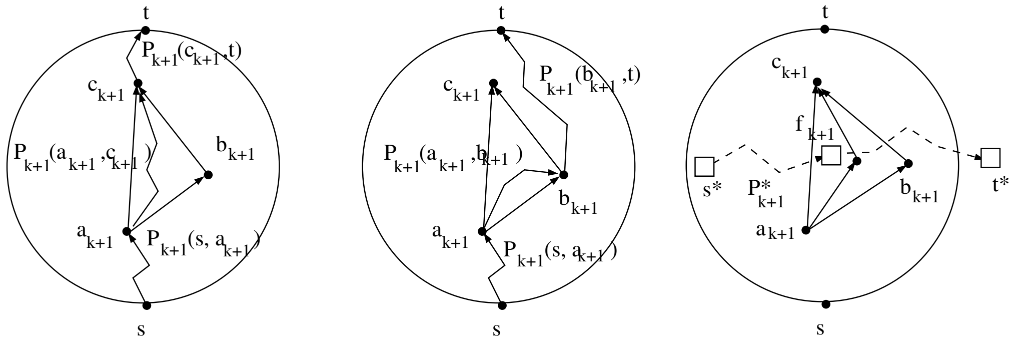
Let $\mathcal{O}_{k+1} = \mathcal{O}_k \oplus \mathcal{O}_{\triangle_{k+1}}$. First we show $\text{length}(\mathcal{O}_{k+1}) \le \frac{2}{3}|G_{k+1}| + \frac{\sum_{i=1}^{k+1} |I(G_{\triangle_i})|}{3} + 14$.

Note that $|G_{k+1}| = |G_k| + |I(G_{\triangle_{k+1}})| = |G_k| + |G_{\triangle_{k+1}}| - 3$.

Let P_{k+1} be a longest path in $\mathcal{O}_{k+1}$ from s to t in G_{k+1}; P_k a longest path in $\mathcal{O}_k$ from s to t in G_k; and $P_{\triangle_{k+1}}$ a longest path in $\mathcal{O}_{\triangle_{k+1}}$ from a_{k+1} to c_{k+1}. There are several cases:

(i) P_{k+1} does not contain any interior edge in $G_{\triangle_{k+1}}$. Then P_{k+1} is a path in G_k. By induction hypothesis,

$$\text{length}(\mathcal{O}_{k+1}) = |P_{k+1}| \le \frac{2}{3}|G_k| + \frac{\sum_{i=1}^{k} |I(G_{\triangle_i})|}{3} + 14 \le \frac{2}{3}|G_{k+1}| + \frac{\sum_{i=1}^{k+1} |I(G_{\triangle_i})|}{3} + 14.$$



(1) (2) (3)

Figure 3 The proof of Theorem 10 (1) Case 2; (2) Case 3; (3) Path in the dual graph.

(ii) P_{k+1} passes through a path in $G_{\triangle_{k+1}}$ from a_{k+1} to c_{k+1} (see Fig. 3 (1)). P_{k+1} can be divided into 3 sub-paths: $\{P_{k+1}(s,a_{k+1}), P_{k+1}(a_{k+1},c_{k+1}), P_{k+1}(c_{k+1},t)\}$. Here $P_{k+1}(s,a_{k+1}), P_{k+1}(c_{k+1},t)$ are paths in G_k. $P_{k+1}(a_{k+1},c_{k+1})$ is a path in $G_{\triangle_{k+1}}$. Since $P_{\triangle_{k+1}}$ is a longest path in $G_{\triangle_{k+1}}$, we have: $|P_{k+1}(a_{k+1},c_{k+1})| \leq |P_{\triangle_{k+1}}|$.
Let P' be the concatenation of: $P_{k+1}(s,a_{k+1})$ followed by the edges $(a_{k+1} \to b_{k+1})$ and $(b_{k+1} \to c_{k+1})$; followed by $P_{k+1}(c_{k+1},t)$. Then P' is a path in G_k. Thus $|P'| = |P_{k+1}(s,a_{k+1})| + 2 + |P_{k+1}(c_{k+1},t)| \leq |P_k|$. This implies: $|P_{k+1}(s,a_{k+1})| + |P_{k+1}(c_{k+1},t)| \leq |P_k| - 2$. Hence:

$$\begin{aligned}
\text{length}(\mathcal{O}_{k+1}) &= |P_{k+1}| = |P_{k+1}(s,a_{k+1})| + |P_{k+1}(a_{k+1},c_{k+1})| + |P_{k+1}(c_{k+1},t)| \\
&\leq |P_k| - 2 + |P_{\triangle_{k+1}}| \leq \frac{2}{3}|G_k| + \frac{\sum_{i=1}^{k}|I(G_{\triangle_i})|}{3} + 14 + |G_{\triangle_{k+1}}| - 1 - 2 \\
&= \frac{2}{3}|G_k| + \frac{\sum_{i=1}^{k}|I(G_{\triangle_i})|}{3} + (|I(G_{\triangle_{k+1}})| + 3) + 14 - 3 \\
&= \frac{2}{3}(|G_k| + |I(G_{\triangle_{k+1}})| + 3) + \frac{\sum_{i=1}^{k+1}|I(G_{\triangle_i})| + 3}{3} + 14 - 3 \\
&= \frac{2}{3}|G_{k+1}| + \frac{\sum_{i=1}^{k+1}|I(G_{\triangle_i})|}{3} + 14
\end{aligned}$$

(iii) P_{k+1} passes through a path in $G_{\triangle_{k+1}}$ from a_{k+1} to b_{k+1} (see Fig. 3 (2)).
P_{k+1} can be divided into three sub-paths: $\{P_{k+1}(s,a_{k+1}), P_{k+1}(a_{k+1},b_{k+1}), P_{k+1}(b_{k+1},t)\}$. Here $P_{k+1}(s,a_{k+1}), P_{k+1}(b_{k+1},t)$ are paths in G_k, while $P_{k+1}(a_{k+1},b_{k+1})$ is a path in $G_{\triangle_{k+1}}$. The concatenation of $P_{k+1}(a_{k+1},b_{k+1})$ and the edge $b_{k+1} \to c_{k+1}$ is a path in $G_{\triangle_{k+1}}$. Hence: $|P_{k+1}(a_{k+1},b_{k+1})| + 1 \leq |P_{\triangle_{k+1}}|$. The concatenation of $P_{k+1}(s,a_{k+1})$ followed by the edge $a_{k+1} \to b_{k+1}$, followed by $P_{k+1}(b_{k+1},t)$ is a path in G_k. So: $|P_{k+1}(s,a_{k+1})| + 1 + |P_{k+1}(b_{k+1},t)| \leq |P_k|$. Hence:

$$\begin{aligned}
\text{length}(\mathcal{O}_{k+1}) &= |P_{k+1}| = |P_{k+1}(s,a_{k+1})| + |P_{k+1}(a_{k+1},b_{k+1})| + |P_{k+1}(b_{k+1},t)| \\
&\leq (|P_k| - 1) + (|P_{\triangle_{k+1}}| - 1) \\
&\leq \frac{2}{3}|G_k| + \frac{\sum_{i=1}^{k}|I(G_{\triangle_i})|}{3} + 14 + |G_{\triangle_{k+1}}| - 3 \\
&= \frac{2}{3}|G_{k+1}| + \frac{\sum_{i=1}^{k+1}|I(G_{\triangle_i})|}{3} + 14
\end{aligned}$$

(iv) P_{k+1} passes through a path in $G_{\triangle_{k+1}}$ from b_{k+1} to c_{k+1}. The proof is symmetric to Case 3.

Next we prove Claim 2. Let P_k^* be a longest path of $\mathcal{O}_k^*$ from s^* to t^*. From induction hypothesis, $|P_k^*| \leq 2|G_k| - 5 - \sum_{i=1}^{k} \mathcal{X}(|I(G_{\triangle_i})|)$. Let $P_{\triangle_{k+1}}^*$ be a longest path in $G_{\triangle_{k+1}}^*$. By Theorem 4, $|P_{\triangle_{k+1}}^*| \leq \lfloor \frac{4}{3}|G_{\triangle_{k+1}}|\rfloor - 2$.

Let P_{k+1}^* be a longest path of $\mathcal{O}_{k+1}^*$ from s^* to t^*. Let f_{k+1} be the face in G_{k+1} that is in the interior of $\triangle_{k+1}$ adjacent to the edge $a_{k+1} \to c_{k+1}$ (see Fig. 3 (3).) (In other words, f_{k+1} corresponds to the source node of dual *st*-orientation of $G_{\triangle_{k+1}}^*$.) If P_{k+1}^* uses any edge in $G_{\triangle_{k+1}}^*$, it must cross the edge $a_{k+1} \to c_{k+1}$ and enter the face f_{k+1}. There are two cases.

(a) P_{k+1}^* does not pass f_{k+1}. Then P_{k+1}^* is a path in G_k^* and the claim trivially holds.
(b) P_{k+1}^* passes through f_{k+1}.

$$\begin{aligned}
\text{length}(\mathcal{O}_{k+1}^*) &= |P_k^*| + |P_{\triangle_{k+1}}^*| - |\{f_{k+1}\}| \leq 2|G_k| - 5 \\
&\qquad - \sum_{i=1}^{k} \mathcal{X}(|I(G_{\triangle_i})|) + \lfloor \frac{4}{3}|G_{\triangle_{k+1}}|\rfloor - 2 - 1 \\
&= 2(|G_{k+1}| - |I(G_{\triangle_{k+1}})|) - 5 \\
&\qquad - \sum_{i=1}^{k} \mathcal{X}(|I(G_{\triangle_i})|) + \lfloor \frac{4}{3}(|I(G_{\triangle_{k+1}})| + 3)\rfloor - 3 \\
&= 2|G_{k+1}| - 5 - \sum_{i=1}^{k} \mathcal{X}(|I(G_{\triangle_i})|) - 2|I(G_{\triangle_{k+1}})| + \lfloor \frac{4}{3}|I(G_{\triangle_{k+1}})|\rfloor + 1 \\
&= 2|G_{k+1}| - 5 - \sum_{i=1}^{k} \mathcal{X}(|I(G_{\triangle_i})|) - (\lceil \frac{2}{3}|I(G_{\triangle_{k+1}}|\rceil - 1) \\
&= 2|G_{k+1}| - 5 - \sum_{i=1}^{k+1} \mathcal{X}(|I(G_{\triangle_i})|)
\end{aligned}$$

◀

▶ **Lemma 11.** *Let $S = \{\triangle_1, \triangle_2, \ldots, \triangle_h\}$ be a set of unrelated separating triangles of G such that $G' = G - (\cup_{i=1}^{h} I(G_{\triangle_i}))$ is a 4-connected graph. Then, G has an st-orientation $\mathcal{O}$ such that* $\text{length}(\mathcal{O}) \leq \frac{3}{4}n + \frac{\sum_{i=1}^{h}|I(G_{\triangle_i})|}{4} + 2\lceil\sqrt{|G'|}\rceil + 4$ *and* $\text{length}(\mathcal{O}^*) \leq \frac{3}{2}n + \frac{\sum_{i=1}^{h}|I(G_{\triangle_i})|}{2}$.

Proof. Define $G_j = G' \cup (\cup_{i=1}^{j} G_{\triangle_i})$. We show, by induction, that G_j has an *st*-orientation $\mathcal{O}_j$ such that

1. $\text{length}(\mathcal{O}_j) \leq \frac{3}{4}n + \frac{\sum_{i=1}^{j}|I(G_{\triangle_i})|}{4} + 2\lceil\sqrt{|G'|}\rceil + 4$
2. $\text{length}(\mathcal{O}_j^*) \leq \frac{3}{2}n + \frac{\sum_{i=1}^{j}|I(G_{\triangle_i})|}{2}$.

Base case $j = 0$: Since $G_0 = G'$ is 4-connected, by Theorem 5, G' has an *st*-orientation $\mathcal{O}'$ such that $\text{length}(\mathcal{O}') \leq \frac{3}{4}|G'| + 2\lceil\sqrt{|G'|}\rceil + 4$ and $\text{length}(\mathcal{O}'^*) \leq \frac{3}{2}|G'|$. The claims are trivially true.

Suppose the claims are true for $j = k$.

Suppose that $\triangle_{k+1} = \{a_{k+1}, b_{k+1}, c_{k+1}\}$. Without loss of generality, assume the edges of $\triangle_{k+1}$ are oriented in $\mathcal{O}_k$ as $\{(a_{k+1} \to b_{k+1}), (b_{k+1} \to c_{k+1}), (a_{k+1} \to c_{k+1})\}$.

By Theorem 2, $G_{\triangle_{k+1}}$ has an *st*-orientation $\mathcal{O}_{\triangle_{k+1}}$, with a_{k+1} as the source and c_{k+1} as the sink, such that $\text{length}(\mathcal{O}_{\triangle_{k+1}}) \leq |G_{\triangle_{k+1}}| - 1$ and $\text{length}(\mathcal{O}_{\triangle_{k+1}}^*) \leq 2|G_{\triangle_{k+1}}| - 5$.

We show the orientation $\mathcal{O}_{k+1} = \mathcal{O}_k \oplus \mathcal{O}_{\triangle_{k+1}}$ satisfies the claims.

The proof of Claim 1 is similar to the first part of the proof of Theorem 10. We only prove Claim 2.

By induction hypothesis, G_k has an *st*-orientation $\mathcal{O}_k$ such that length$(\mathcal{O}^*_k) \le \frac{3}{2}|G_k| + \frac{\sum_{i=1}^{k}|I(G_{\triangle_i})|}{2}$. Also, we know that length$(\mathcal{O}^*_{\triangle_{k+1}}) \le 2|G_{\triangle_{k+1}}|-5$. As in the proof of Theorem 10, there are two cases for analyzing length$(\mathcal{O}^*_{k+1})$.

(a) P^*_{k+1} does not pass f_{k+1}. Then P^*_{k+1} is a path in G^*_k and the claim trivially holds.

(b) P^*_{k+1} passes f_{k+1}. Then:

$$\begin{aligned}\text{length}(\mathcal{O}) &\le \frac{3}{2}|G_k| + \frac{1}{2}\sum_{i=1}^{k}|I(G_{\triangle_i})| + 2|G_{\triangle_{k+1}}| - 5 - 1\\ &= \frac{3}{2}|G_k| + \frac{1}{2}\sum_{i=1}^{k}|I(G_{\triangle_i})| + 2|I(G_{\triangle_{k+1}})| \le \frac{3}{2}|G_{k+1}| + \frac{1}{2}\sum_{i=1}^{k+1}|I(G_{\triangle_i})|\end{aligned}$$

This completes the induction. ◀

▶ **Theorem 12.** *Let G_v be a 4-block component of G, with the corresponding separating triangle $\triangle_v$ in G. Then G has an st-orientation $\mathcal{O}$ such that* length$(\mathcal{O}) \le \frac{3}{4}n + \frac{1}{4}(n - |G_v|) + 2\lceil\sqrt{|G_v|}\rceil + 4$ *and* length$(\mathcal{O}^*) \le \frac{3}{2}n + \frac{n-|G_v|}{2}$.

Proof. Let $S = \{\triangle_1, \triangle_2, \ldots, \triangle_h\}$ be the set of maximal separating triangles of $G_{\triangle_v}$. Since G_v is 4-connected, by Lemma 11, $G_{\triangle_v}$ has an *st*-orientation $\mathcal{O}_{\triangle_v}$ such that:

$$\begin{aligned}\text{length}(\mathcal{O}_{\triangle_v}) &\le \frac{3}{4}|G_{\triangle_v}| + 2\lceil\sqrt{|G_v|}\rceil + 4 + \frac{\sum_{i=1}^{h}|I(G_{\triangle_i})|}{4}\\ \text{length}(\mathcal{O}^*_{\triangle_v}) &\le \frac{3}{2}|G_{\triangle_v}| + \frac{\sum_{i=1}^{h}|I(G_{\triangle_i})|}{2}.\end{aligned}$$

Let $G_{ext} = G - I(G_{\triangle_v})$. Then G_{ext} has an *st*-orientation such that length$(\mathcal{O}_{ext}) \le |G_{_{ext}}| - 1$ and length$(\mathcal{O}^*_{ext}) \le 2|G_{ext}| - 5$. Let $\mathcal{O} = \mathcal{O}_{ext} \oplus \mathcal{O}_{\triangle_v}$. Then:

$$\begin{aligned}\text{length}(\mathcal{O}) &\le \text{length}(\mathcal{O}_{ext}) + \text{length}(\mathcal{O}_{\triangle_v}) - 2\\ &\le (|G_{_{ext}}| - 1) + \frac{3}{4}|G_{\triangle_v}| + 2\lceil\sqrt{|G_v|}\rceil + 4 + \frac{\sum_{i=1}^{h}|I(G_{\triangle_i})|}{4} - 2\\ &= \frac{3}{4}|G_{ext}| + \frac{1}{4}|G_{ext}| + \frac{3}{4}|G_{\triangle_v}| + 2\lceil\sqrt{|G_v|}\rceil + \frac{\sum_{i=1}^{h}|I(G_{\triangle_i})|}{4} + 1\\ &= \frac{3}{4}(|G| + 3) + \frac{1}{4}(|V(G_{ext}) \cup (\cup_{i=1}^{h} I(G_{\triangle_i}))|) + 2\lceil\sqrt{|G_v|}\rceil + 1\\ &= \frac{3}{4}(n + 3) + \frac{1}{4}(n - |G_v| + 3) + 2\lceil\sqrt{|G_v|}\rceil + 1\\ &= \frac{3}{4}n + \frac{1}{4}(n - |G_v|) + 2\lceil\sqrt{|G_v|}\rceil + 4\\ \text{length}(\mathcal{O}^*) &= \text{length}(\mathcal{O}{*}_{ext}) + \text{length}(\mathcal{O}^*_{\triangle_v}) - 1\\ &\le (2|G_{ext}| - 5) + (\frac{3}{2}|G_{\triangle_v}| + \frac{\sum_{i=1}^{h}|I(G_{\triangle_i})|}{2}) - 1\\ &= \frac{3}{2}|G_{ext}| + \frac{3}{2}|G_{\triangle_v}| + \frac{1}{2}|G_{ext}| + \frac{\sum_{i=1}^{h}|I(G_{\triangle_i})|}{2} - 6\\ &= \frac{3}{2}(|G| + 3) + \frac{1}{2}(|I(G_{ext}) \cup (\cup_{i=1}^{h} I(G_{\triangle_i}))| + 3) - 6 = \frac{3}{2}n + \frac{1}{2}(n - |G_v|)\end{aligned}$$

This completes the proof. ◀

▶ **Theorem 13.** *Every plane triangulation G of n vertices has a VR with height* $\leq \max\{\frac{23}{24}n + 2\lceil\sqrt{n}\rceil + 4, \frac{11}{12}n + 13\}$ *and width* $\leq \frac{23}{12}n$.

Proof. By Lemma 9, there are two cases:

Case 1: G has a 4-block component with size $n_1 \geq n/6$. By Theorem 12, G has an st-orientation $\mathcal{O}$ such that $\text{length}(\mathcal{O}) \leq \frac{3}{4}n + \frac{n-n_1}{4} + 2\lceil\sqrt{n}\rceil + 4$ and $\text{length}(\mathcal{O}^*) \leq \frac{3n}{2} + \frac{(n-n_1)}{2}$. Since $n_1 \geq \frac{n}{6}$, we have: $\text{length}(\mathcal{O}) \leq \frac{23}{24}n + 2\lceil\sqrt{n}\rceil + 4$ and $\text{length}(\mathcal{O}^*) \leq \frac{23}{12}n$.

Case 2: G has a set of unrelated separating triangles $\{\triangle_1, \triangle_2, \ldots, \triangle_h\}$ such that:

- For all i, $|G_{\triangle_i}| \geq 5$, (which implies $|I(G_{\triangle_i})| \geq 2$).
- $\frac{n}{4} - 3 \leq \sum_{i=1}^{h} |I(G_{\triangle_i})| \leq \frac{3}{4}n - 3$.

Since $\mathcal{X}(z) \geq z/3$ for all $z \geq 2$, we have:

$$\sum_{i=1}^{h} \mathcal{X}(|I(G_{\triangle_i})|) \geq \frac{\sum_{i=1}^{h} |I(G_{\triangle_i})|}{3}.$$

By Theorem 10, G has an st-orientation $\mathcal{O}$ such that

$$\begin{aligned}
\text{length}(\mathcal{O}) &\leq \frac{2n}{3} + \frac{|\cup_{i=1}^{h} I(G_{\triangle_i})|}{3} + 14 \leq \frac{2n}{3} + \frac{3n/4 - 3}{3} + 14 = \frac{11}{12}n + 13 \\
\text{length}(\mathcal{O}^*) &\leq 2n - 5 - \sum_{i=1}^{h} \mathcal{X}(|I(G_{\triangle_i})|) \leq 2n - 5 - \frac{n/4 - 3}{3} < \frac{23}{12}n.
\end{aligned}$$

In either case, the orientation $\mathcal{O}$ leads to a VR of G with the stated width and height. ◀

5 Conclusion

In this paper, we showed that every plane graph of n vertices has a VR with height $\leq \max\{\frac{23}{24}n + 2\lceil\sqrt{n}\rceil + 4, \frac{11}{12}n + 13\}$ and width $\leq \frac{23}{12}n$. This is the first VR construction for general plane graphs that simultaneously bounds the height and the width from the trivial upper bound. The gap between the size of our VR and the known lower bound is still large. It would be interesting to find more compact VR constructions.

References

1 G. di Battista, P. Eades, R. Tammassia, and I. Tollis, Graph Drawing: Algorithms for the Visualization of Graphs, Prentice Hall, 1999.

2 C.Y. Chen and Y.F. Hung and H.I. Lu, Visibility Representations of Four Connected Plane Graphs with Near Optimal Heights, *GD 2009*, LNCS, vol. 5417, pp. 67-77, Springer, 2009.

3 J.H. Fan, C.C. Lin, H.I. Lu and H.C. Yen, Width-optimal visibility representations of plane graphs, *ISSAC 2007*, LNCS vol. 4835, pp. 160-171, Springer, 2007.

4 H. Zhang and X. He, Nearly optimal visibility representations on plane graphs, *ICALP 2006*, LNCS, vol. 4051, pp. 407-418, Springer, 2006.

5 X. He, J-J Wang, and H. Zhang Compact Visibility Representation of 4-Connected Plane Graphs, to appear in Proc. *COCOA 2010*.

6 G. Kant, A more compact visibility representation. *International Journal of Computational Geometry and Applications* 7, pp. 197-210, 1997.

7 G. Kant and X. He, Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theoretical Computer Science* 172, pp. 175-193, 1997.

8 A. Lempel, S. Even and I. Cederbaum, An algorithm for planarity testing of graphs, in *Theory of Graphs (Proc. of an International Symposium, Rome,July 1966)*, pp. 215-232, 1967.

9 C.-C. Lin, H.-I. Lu and I-F. Sun, Improved compact visibility representation of planar graph via Schnyder's realizer, *SIAM Journal on Discrete Mathematics*, 18, pp. 19-29, 2004.

10 P. Ossona de Mendez, Orientations bipolaires. PhD thesis, Ecole des Hautes Etudes en Sciences Sociales, Paris, 1994.

11 Charalampos Papamanthou and Ioannis G. Tollis, Applications of Parameterized st-Orientations in Graph Drawing Algorithms, *Proc. 13th International Symposium on Graph Drawing*, LNCS vol. 3843, pp. 355-367, Springer, 2005.

12 Charalampos Papamanthou and Ioannis G. Tollis, Parameterized st -Orientations of Graphs: Algorithms and Experiments, *Proc. 14th International Symposium on Graph Drawing*, LNCS Vol. 4372, pp. 220-233, Springer, 2006.

13 P. Rosenstiehl and R. E. Tarjan, Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete Comput. Geom.* 1, pp. 343-353, 1986.

14 R. Tamassia and I.G.Tollis, An unified approach to visibility representations of planar graphs. *Discrete Comput. Geom.* 1, pp. 321-341, 1986.

15 H. Zhang and X. He, Canonical Ordering Trees and Their Applications in Graph Drawing, *Discrete Comput. Geom.* 33, pp. 321-344, 2005.

16 H. Zhang and X. He, Visibility Representation of Plane Graphs via Canonical Ordering Tree, *Information Processing Letters* 96, pp. 41-48, 2005.

17 H. Zhang and X. He, Improved Visibility Representation of Plane Graphs, *Computational Geometry: Theory and Applications* 30, pp. 29-29, 2005.

18 H. Zhang and X. He, An Application of Well-Orderly Trees in Graph Drawing, *Proc. 13th International Symposium on Graph Drawing*, LNCS Vol. 3843, pp. 458-467, Springer, 2005.

19 H. Zhang and X. He, Optimal *st*-orientations for plane triangulations, *J. Comb. Optim.* 17, pp. 367-377, 2009.

Telling convex from reflex allows to map a polygon

Jérémie Chalopin*[1], Shantanu Das[1], Yann Disser[2], Matúš Mihalák[2], and Peter Widmayer[2]

1 **LIF, CNRS & Aix-Marseille Université, France**
2 **Institute of Theoretical Computer Science, ETH Zürich, Switzerland**

Abstract

We consider the exploration of a simple polygon $\mathcal{P}$ by a robot that moves from vertex to vertex along edges of the visibility graph of $\mathcal{P}$. The visibility graph has a vertex for every vertex of $\mathcal{P}$ and an edge between two vertices if they see each other, i.e. if the line segment connecting them lies inside $\mathcal{P}$ entirely. While located at a vertex, the robot is capable of ordering the vertices it sees in counter-clockwise order as they appear on the boundary, and for every two such vertices, it can distinguish whether the angle between them is convex ($\leq \pi$) or reflex ($> \pi$). Other than that, distant vertices are indistinguishable to the robot. We assume that an upper bound on the number of vertices is known and show that the robot is always capable of reconstructing the visibility graph of $\mathcal{P}$. We also show that multiple identical, indistinguishable and deterministic such robots can always position themselves such that they mutually see each other, i.e. such that they form a clique in the visibility graph.

1998 ACM Subject Classification F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems — Computations on discrete structures, Geometrical problems and computations; G.2.2 [Discrete Mathematics]: Graph Theory — Visibility graphs

Keywords and phrases polygon mapping, map construction, autonomous agent, simple robot, visibility graph reconstruction

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.153

1 Introduction

Autonomous mobile robots are used for various tasks like cleaning, guarding, data retrieval, etc. in unknown environments. Many tasks require the exploration of the environment and the creation of a map. The difficulty of the mapping problem depends on the characteristics of the environment itself and on the sophistication of the robots, i.e. on their sensory and locomotive capabilities. A natural question is how much sophistication a robot needs to be able to solve the problem. The ultimate goal is to characterize the difficulty of the mapping problem by finding minimal robot configurations that allow a robot to create a map.

We focus on robots operating in simple polygons. For many tasks, instead of inferring a detailed map of the geometry of the environment, it is enough to obtain the visibility graph. The visibility graph has a node for each vertex of the polygon and an edge connecting two nodes if the corresponding vertices *see each other*, i.e. if the straight-line segment between them is contained in the polygon. The goal in this context becomes to find minimal robot models that allow a robot inside a polygonal environment to reconstruct the visibility graph of its environment. The information the robot can gather must be sufficient to *uniquely* infer the visibility graph.

[*] Partially supported by ANR Projects SHAMAN and ECSPER.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 153–164
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

A variety of minimalistic robot models have been studied, focusing on different types of environments and objectives [1, 5, 8, 10, 14]. The model considered here originates from [12]. Roughly speaking, our robot is allowed to move along edges of the visibility graph. While at a vertex, the robot sees the vertices visible to its current location in counter-clockwise (ccw) order starting with its ccw neighbor along the boundary. Apart from this ordering the vertices are indistinguishable to the robot. In each move the robot may select one of them and move to it. The robot has no way of *looking back*, i.e. it has no immediate way of knowing which vertex it came from among the vertices it sees now. However, the robot is assumed to be aware of is an upper bound $\bar{n}$ on the number of vertices n.

Unless extended with additional capabilities, a robot as defined above cannot reconstruct the visibility graph of a polygon when restricted to moving along the boundary only [3]. If we allow the robot to measure the angles between pairs of visible vertices in addition to ordering them, moving along the boundary is sufficient to reconstruct the visibility graph however [9]. As soon as a robot starts moving across the polygon (as opposed to along the boundary), the lack of the ability to look back makes it difficult for the robot to relate the information it collected so far to subsequent observations. It thus makes sense to consider *look-back robots* which have the ability to look back and identify the vertex they came from in their last move. This ability empowers a look-back robot to retrace all of its movements. If we add the ability to distinguish convex ($\leq \pi$) and reflex ($> \pi$) angles, it was shown that a look-back robot can reconstruct the visibility graph [3]. Later, it was shown that a look-back robot in fact does not even need to distinguish convex and reflex angles [6]. In the same paper, it was also shown that look-back robots can solve the weak-rendezvous problem in which multiple identical, indistinguishable and deterministic robots need to position themselves such that they mutually see each other. In the following we show that a robot can reconstruct the visibility graph even without looking back, as long as it can distinguish convex and reflex angles. Along the way, we show that such robots can also solve the weak-rendezvous problem.

In the robot model we use, robots move along edges of the visibility graph and can locally access some information about the edges. We can model this in the context of general robotic exploration of edge-labeled graphs, where the edge-labeling is usually restricted to be locally bijective at every vertex (i.e. no two edges incident to the same vertex have the same label). In this more general context, robots are aware of the degree of the vertex they are located at as well as of the labels of the edges incident to it. In every step, the robot selects an edge and move to its other end. It is known that labeled graphs can appear mutually indistinguishable to a robot, i.e. the reconstruction problem is not always solvable [2, 4]. The rendezvous problem is generally not solvable either [7, 13]. We will see later, that polygon exploration can be transformed to the exploration of a particular class of directed, arc-labeled graphs, where both the reconstruction problem as well as the weak-rendezvous problem are solvable.

As it is impossible to reconstruct general graphs, it is natural to ask how much information a robot can obtain about a graph. This information is encoded in the unique *minimum base graph* of a graph G – the smallest graph among all graphs indistinguishable from G by a robot [4]. In general, the mapping from a graph to its minimum base graph is not one-to-one in the sense that there are graphs which share the same minimum base graph. Our question whether a robot with certain capabilities can reconstruct the visibility graph of a polygon can be translated to whether the mapping is one-to-one for the class of visibility graphs with an appropriate labeling. We show that if the labeling locally encodes the convexity information about every angle at a vertex, this mapping becomes one-to-one. In other words, visibility graphs can be reconstructed from their minimum base graph if a bound $\bar{n}$ on the total number of vertices and the type of every angle (convex or reflex) are known.

2 The visibility graph reconstruction problem

We consider the exploration of a (simple) polygon $\mathcal{P}$ by a robot that moves from vertex to vertex along straight lines in $\mathcal{P}$. Two vertices u, v that can be connected with a straight line inside $\mathcal{P}$ are said to *see each other*. We define the visibility graph $G_{\text{vis}} = (V, E)$ of $\mathcal{P}$ to be a directed graph, where V is the set of vertices of $\mathcal{P}$ and there is an arc from u to v (and vice versa) if u and v see each other. Whenever convenient we identify G_{vis} with its canonical straight-line embedding in the polygon. For example, we speak of angles between arcs of G_{vis} meaning the angles between the corresponding line segments of its straight-line embedding.

Depending on the additional capabilities we equip a robot with, it might or might not be able to perform certain tasks. We focus on the *visibility graph reconstruction problem* in which the robot has to uniquely infer G_{vis}. Here and throughout this paper we consider isomorphic graphs to be the "same" graph, as we cannot hope to distinguish graphs further. We also consider the *weak-rendezvous problem* in which multiple identical and deterministic robots need to position themselves on vertices of the polygon that mutually see each other.

Before defining a specific robot model we introduce some formalism for G_{vis}. We fix a vertex v_0 and denote the vertices of $\mathcal{P}$ in ccw order along the boundary by $v_0, v_1, \ldots, v_{n-1}$. Note that $v_0, v_1, \ldots, v_{n-1}, v_0$ is a Hamiltonian cycle in G_{vis}. By $\text{chain}(v_l, v_r)$ we denote the sequence $(v_l, v_{l+1}, \ldots, v_r)$ and by $\text{chain}_v(v_l, v_r)$ we denote the subsequence of $\text{chain}(v_l, v_r)$ containing only the vertices visible to v. Here and throughout this paper all indices are understood modulo n. Let $v_i \in V$ and $(u_1, \ldots, u_{d_i}) := \text{chain}_{v_i}(v_{i+1}, v_{i-1})$ be the vertices visible to v_i. We say d_i is the degree of v_i and define $\text{vis}_{v_i}(x) := \text{vis}_{v_i}(-(d_i + 1 - x)) := u_x$ to be the x-th vertex visible to v_i in ccw order or equivalently the $(d_i + 1 - x)$-th vertex visible to v_i in clockwise (cw) order for $1 \leq x \leq d_i$. Conversely, we set $O_{v_i}(u_x) := x$ or interchangeably $O_{v_i}(u_x) = -(d_i + 1 - x)$ for $1 \leq x \leq d_i$. For $1 \leq x < y \leq d_i$ we write $A_{v_i}(x, y) = A_{v_i}(y, x)$ to denote the ccw angle between the arcs (v_i, u_x) and (v_i, u_y) in that order. Furthermore, we define the *angle type* $T_{v_i}(\cdot, \cdot)$ as follows: $T_{v_i}(x, y) = T_{v_i}(y, x) = 1$ if $A_{v_i}(x, y) > \pi$ and $T_{v_i}(x, y) = 0$ otherwise. For convenience we set $T_{v_i}(x, x) = 0$. A vertex v_i is called reflex if $T_{v_i}(1, d_i) = 1$ and convex otherwise.

The exploration of G_{vis} can be reduced to the general problem of exploring a strongly connected, directed and arc-labeled graph G (from now on we use the word "graph" to refer to such graphs). We write $\lambda(e)$ to denote the label of an arc e. A robot exploring a graph is assumed to be aware of the labels of all the outgoing arcs at its location. In every move, the robot may choose one of those arcs and follow it to its target. In the following we distinguish between (directed) paths that visit every vertex at most once and (directed) walks that do not have this restriction. Every walk p in the graph uniquely induces a label-sequence $\lambda(p)$. Conversely, any label-sequence Λ induces a set of walks $\Lambda(G)$ such that $\lambda(p) = \Lambda$ for all $p \in \Lambda(G)$. By $\Lambda(v)$ we denote the set of walks in $\Lambda(G)$ that start at v. If no two outgoing arcs of the same vertex share a label, we say the graph has a *local orientation* or is *locally oriented*. Then, for every label-sequence Λ and vertex v we have $\Lambda(v) = \emptyset$ or $|\Lambda(v)| = 1$; in the latter case we write $\Lambda(v)$ to denote this unique walk.

We can now introduce our robot model in detail. As described above, we allow a robot to move along arcs of the visibility graph. In addition, while situated at a vertex v of degree d, the robot can order all outgoing arcs in ccw order starting with the arc to its ccw neighbor along the boundary, and is aware of $T_v(x, y)$ for all $1 \leq x, y \leq d$. We assume the robot to be aware of an upper bound $\bar{n} \geq n$ on the total number of vertices n. From now on, when we talk about a robot in a polygon, we refer to the robot model described above.

The exploration of $\mathcal{P}$ by a robot is in fact equivalent to the exploration of an arc-labeled version of G_{vis}, with an appropriate labeling that encodes the information available to the robot. For every vertex of the polygon we need to encode the local orientation and the angle type information into a labeling of the outgoing arcs at the corresponding vertex in G_{vis}. We introduce a labeling in which each label is a sequence of integers. Let u be a vertex of the visibility graph with degree d and (u, v) be an outgoing arc of u. We label (u, v) with the label $(x_0, x_1, \ldots, x_d)$, where $x_0 := O_u(v)$ and $x_i := T_u(x_0, i)$. Note that by the definition of O_u our labeling is a local orientation. Further note that the arcs (u, v) and (v, u) can be labeled differently. It is immediate to check that a robot exploring G_{vis} encounters the exact same information as a robot inside the polygon (that is aware of T_v) if both start at the same vertex. It is thus sufficient to show that the labeled graph G_{vis} can be reconstructed in the framework of exploring general graphs in order to show that a robot can indeed solve the visibility graph reconstruction problem.

3 Overview of the algorithm

The visibility graph reconstruction algorithm that we design in this paper combines several old and new graph-theoretical and geometrical properties of visibility graphs as well as techniques developed in earlier studies. Rather than formally introducing all relevant concepts right away, this section aims to give an intuitive overview of the algorithm. We informally describe the underlying techniques and defer their formal discussion to later sections. Note that we are primarily interested in showing that a robot is at all capable of uniquely reconstructing the visibility graph of any simple polygon. The algorithm we provide as a proof does not need to be particularly efficient as long as it is guaranteed to terminate in finite time. An algorithm that solves the weak-rendezvous problem is obtained as a byproduct.

In Section 2 we argued that the exploration of $\mathcal{P}$ by a robot is equivalent to the exploration of G_{vis} in the context of general graph exploration. In general and without any prior knowledge of the graph, there can be infinitely many graphs that are compatible with the observations of the robot no matter how far it moves, i.e. all these graphs are indistinguishable to the robot. However, it is known [4] that for every graph G, there is always a unique *minimum base graph* $G^\star$ that is indistinguishable from G and has minimum size. Using the fact that G_{vis} is locally oriented and that an upper bound $\bar{n}$ on n is known a priori, we are able to show the following result.

▶ **Theorem 1.** *A robot in $\mathcal{P}$ can determine $G^\star_{\mathrm{vis}}$.*

The main ingredient for this theorem is the observation that given two candidate graphs for $G^\star_{\mathrm{vis}}$, the robot can eliminate one of them in finite time by following an appropriate sequence of arc labels. It is then sufficient to iterate over pairs of graphs with size at most $\bar{n}$, discarding one of the two in every step. Once the robot determined $G^\star_{\mathrm{vis}}$, it has extracted all the information it can possibly gather by moving around. Subsequent steps of the algorithm can thus operate on $G^\star_{\mathrm{vis}}$ directly without further need of moving at all in G_{vis}.

We associate each vertex of G_{vis} with a vertex of $G^\star_{\mathrm{vis}}$ such that each vertex of $G^\star_{\mathrm{vis}}$ represents a *class* of vertices of G_{vis}. For two vertices u, v of G_{vis} in the same class we have $\Lambda(u) = \emptyset \Leftrightarrow \Lambda(v) = \emptyset$ for all label-sequences Λ. Furthermore, the classes with which the vertices are associated repeat periodically along the boundary and in particular all classes have the same size. We define a unique order between the classes and use a procedure similar to the one in [6] to show that at least one of them forms a clique in G_{vis}. The idea is to repeatedly "cut off" *ears* of the polygon, i.e. vertices whose neighbors on the boundary see each other. Cutting off such an ear yields a subpolygon of $\mathcal{P}$ and we can repeat the process

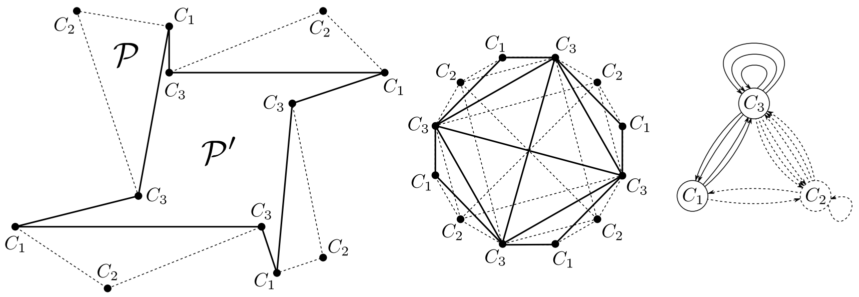


Figure 1 Left: cutting away a class of vertices (ears) from $\mathcal{P}$ to obtain $\mathcal{P}'$. Middle: visibility graph G_{vis} of $\mathcal{P}$. Right: minimum base graph $G^{\star}_{\text{vis}}$ of G_{vis}. Dashed edges are in $\mathcal{P}$ but not in $\mathcal{P}'$.

on the subpolygon. However, the robot cannot operate on G_{vis} directly as it only has access to $G^{\star}_{\text{vis}}$. The following lemma allows the robot to cut off an entire class of vertices at a time, an operation that can be performed in $G^{\star}_{\text{vis}}$ simply by deleting the corresponding vertex (and adjusting the arc labels of its neighboring vertices).

▶ **Lemma 2.** *Let v be an ear of $\mathcal{P}$. Then every vertex in the same class as v is an ear of $\mathcal{P}$.*

As every polygon has at least one ear, the robot can thus "cut off" an entire class of $\mathcal{P}$ in order to obtain a new and smaller polygon $\mathcal{P}'$ (cf. Figure 1). By removing the corresponding vertex of $G^{\star}_{\text{vis}}$ and updating the arc labels, it obtains a graph $G'^{\star}_{\text{vis}}$ that is indistinguishable from the visibility graph of $\mathcal{P}'$. If this process is repeated, always selecting the smallest class with respect to the order relation for removal, eventually a situation is reached in which only one (uniquely defined) class $C^{\star}$ remains. As the corresponding subpolygon must again have at least one ear, by the above lemma the entire class $C^{\star}$ consists of ears and the corresponding subpolygon thus is convex. A convex subpolygon is a clique in the original visibility graph and we may conclude the following crucial theorem.

▶ **Theorem 3.** *There is a uniquely defined class $C^{\star}$ in G_{vis} whose vertices form a clique.*

While the robot could explicitly execute the procedure described above, finding the class $C^{\star}$ can be done much more directly. If the number of self-loops of a vertex in $G^{\star}_{\text{vis}}$ equals the size of the corresponding class minus one, this class is a clique. It is thus enough to inspect all classes in turn. Among all classes that form a clique, the largest class with respect to the order relation must be $C^{\star}$. The previous theorem guarantees the existence of such a class. This result also gives a robot the means to infer n from $\bar{n}$: n is equal the size of $C^{\star}$ times the number of classes in G_{vis}. To compute the size of $C^{\star}$, the robot can do the following. Consider a vertex v in G^{*}_{vis} such that the number of self-loops incident to v is greater or equal than the number of self-loops incident to any other vertex of G^{*}_{vis}. Then, the class C corresponding to v is a clique and there are exactly $|C| - 1$ self-loops incident to v.

The above yields an algorithm for multiple robots to weakly meet: As $C^{\star}$ is unique, every robot can determine $C^{\star}$ and then simply position itself on a vertex of $C^{\star}$. We get

▶ **Theorem 4.** *Any number of robots in $\mathcal{P}$ can solve the weak-rendezvous problem.*

Starting from the clique $C^{\star}$, we show that by sequentially "gluing" ears back to the polygon, a robot can extend the initial clique and reconstruct the entire visibility graph step by step. Every step relies on a recursive counting method that was first introduced in [3]. In order to know how to glue ears back on, the robot explicitly needs to construct $C^{\star}$ by repeatedly cutting off ears and remember in which order the classes are cut off in the process.

▶ **Theorem 5.** *A robot in $\mathcal{P}$ can solve the visibility graph reconstruction problem.*

4 Finding the minimum base graph $G^{\star}_{\text{vis}}$

This section focuses on the problem of exploring a general, locally oriented directed graph $G = (V, E)$ with a robot. Again, we assume an upper bound $\bar{n}$ on the number of vertices n to be known and we do not impose a limitation on the memory of the robot. We prove a generalization of Theorem 1 to general, locally oriented graphs.

Before we define the notion of the *minimum base graph* $G^{\star}$ of G we need to introduce a few graph-theoretical concepts. First, given an arc e from vertex u to vertex v, we denote by $s(e)$ the *source of arc* e, i.e. the vertex u, and by $t(e)$ the *target of arc* e, i.e. the vertex v. Note that in the following we allow graphs to have parallel arcs between a pair of vertices. A *morphism* $\mu : G \to G'$ from G to a graph G' is a mapping from G to G' that maps vertices to vertices and arcs to arcs and maintains adjacencies and arc labels. More formally, if e is an arc in G from u to v then $s(\mu(e)) = \mu(u)$, $t(\mu(e)) = \mu(v)$, and $\lambda(e) = \lambda(\mu(e))$. An *opfibration* $\varphi : G \to \bar{G}$ with $\bar{G} = \left(\bar{V}, \bar{E}\right)$ is a morphism such that for every arc $\bar{e} \in \bar{E}$ with $\bar{u} = s(\bar{e})$ and for every vertex $u \in \varphi^{-1}(\bar{u})$ in the preimage of $\bar{u}$ there is a *unique* arc e with source $s(e) = u$ such that $\varphi(e) = \bar{e}$. We say that $\bar{G}$ is a *base graph* of G and G is a *total graph* of $\bar{G}$. Trivially, G is both its own base graph and total graph. If G has no base graph smaller than itself, we say G is *opfibration prime*. An *out-tree* is a graph that has a *root* vertex r such that there is exactly one directed walk from r to every other node.

We give the following properties without proof. For a detailed discussion, refer to [4].

▶ **Proposition 6.** *Let $\varphi : G \to \bar{G}$ be an opfibration. For every label-sequence Λ and every vertex $v \in V$ we have that $\Lambda(v) \neq \emptyset$ iff $\Lambda(\varphi(v)) \neq \emptyset$.*

▶ **Proposition 7.** *There is exactly one opfibration prime base graph of G. We call it the* minimum base graph *of G and denote it with $G^{\star}$.*

▶ **Proposition 8.** *For every $v \in V$, there is a unique (but not necessarily finite) total graph H_v of G that is an out-tree with root in $\varphi^{-1}(v)$, where φ is the opfibration mapping H_v to G. We call it the* universal total graph *of G at v.*

▶ **Proposition 9.** *A graph is opfibration prime iff all its universal total graphs are distinct.*

▶ **Proposition 10.** *Two different opfibration prime graphs have different sets of universal total graphs.*

We can now show that if we have a local orientation, there is a label sequence of finite length that can be used to distinguish any two opfibration prime graphs.

▶ **Lemma 11.** *Let $G_1 = (V_1, E_1), G_2 = (V_2, E_2)$ be two distinct, locally oriented opfibration prime graphs. There is a label-sequence δ of finite length for which $\delta(G_1) = \emptyset$ and $\delta(G_2) \neq \emptyset$ or vice versa.*

Proof. We first show that (without loss of generality) there is a vertex $x \in V_1$ such that for every vertex $v_2 \in V_2$ there is a label-sequence δ_{x,v_2} of finite length with $\delta_{x,v_2}(x) \neq \emptyset$ and $\delta_{x,v_2}(v_2) = \emptyset$ or vice versa. By Proposition 10, without loss of generality, there is a vertex $x \in V_1$ such that the universal total graph H_x of G_1 at x is not a total graph of G_2. Then for every vertex $v_2 \in V_2$, H_x and the universal total graph H_{v_2} of G_2 at v_2 are different. Because G_1 and G_2 are locally oriented, so are H_x and H_{v_2}. Let r_x and r_{v_2} be the roots of H_x and H_{v_2} respectively. Because H_x, H_{v_2} are distinct and locally oriented, there is a finite label-sequence δ_{x,v_2} with $\delta_{x,v_2}(r_x) \neq \emptyset$ and $\delta_{x,v_2}(r_{v_2}) = \emptyset$ or vice versa. By Proposition 6 this implies $\delta_{x,v_2}(x) \neq \emptyset$ and $\delta_{x,v_2}(v_2) = \emptyset$ or vice versa.

We now describe how to use the above to obtain the desired label-sequence δ. We start with the empty label-sequence $\delta^{(0)}$ and iteratively extend it to a longer but still finite label-sequence $\delta^{(i)}$ in step i. Let $A^{(i)} := \{v \in V_1 | \delta^{(i)}(v) \neq \emptyset\}$ and $B^{(i)} := \{v \in V_2 | \delta^{(i)}(v) \neq \emptyset\}$ be the sets of vertices that are "compatible" with $\delta^{(i)}$. As $\delta^{(i+1)}$ extends $\delta^{(i)}$, we have by construction that $A^{(i+1)} \subseteq A^{(i)}$ and $B^{(i+1)} \subseteq B^{(i)}$. We show that our extension satisfies $\left(A^{(i+1)} \cup B^{(i+1)}\right) \subsetneq \left(A^{(i)} \cup B^{(i)}\right)$ in every step and that either $\delta^{(i+1)}(G_1) \neq \emptyset$ or $\delta^{(i+1)}(G_2) \neq \emptyset$. At some point we thus obtain a label-sequence δ for which exactly one graph has no compatible vertices. It remains to show the existence of such an extension.

Let $\delta^{(i)}$ be a finite label-sequence with $\delta^{(i)}(G_1) \neq \emptyset$ or $\delta^{(i)}(G_2) \neq \emptyset$. If $A^{(i)} = \emptyset$ or $B^{(i)} = \emptyset$, we have either $\delta^{(i)}(G_1) = \emptyset$ or $\delta^{(i)}(G_2) = \emptyset$. We can thus set $\delta = \delta^{(i)}$ and are done. So assume $A^{(i)} \neq \emptyset$ and $B^{(i)} \neq \emptyset$. Then, there are two vertices $v_1 \in A, v_2 \in B$. Let $p_1 = \delta^{(i)}(v_1)$, $p_2 = \delta^{(i)}(v_2)$ and v_1' be the target of p_1 (i.e. the vertex at which p_1 ends). As G_1 is strongly connected, there is a path q from v_1' to x, where x is defined as above. Let $\pi = \lambda(q)$ be the associated label-sequence and $\pi^+ = \delta^{(i)} \circ \pi$, where "$\circ$" denotes the concatenation of sequences. We certainly have $\pi^+(v_1) \neq \emptyset$ and thus $\pi^+(G_1) \neq \emptyset$. If $\pi^+(v_2) = \emptyset$, we set $\delta^{(i+1)} = \pi^+$ and have $B^{(i+1)} \subsetneq B^{(i)}$. Otherwise let v_2'' be the target of $\pi^+(v_2)$ (remember that x is the target of $\pi^+(v_1)$). Without loss of generality, we can set $\delta^{(i+1)} = \pi^+ \circ \delta_{x,v_2''}$. By definition of $\delta_{x,v_2''}$, we have $\delta^{(i+1)}(v_1) \neq \emptyset$ and $\delta^{(i+1)}(v_2) = \emptyset$ or vice versa. Thus $A^{(i+1)} \subsetneq A^{(i)}$ or $B^{(i+1)} \subsetneq B^{(i)}$ and hence $\left(A^{(i+1)} \cup B^{(i+1)}\right) \subsetneq \left(A^{(i)} \cup B^{(i)}\right)$. ◀

▶ **Theorem 12.** *A robot exploring G can determine $G^\star$ if it knows an upper bound $\bar{n}$ on the size of G.*

Proof. We prove the theorem for the case when the robot knows n exactly and show later how to generalize the approach to the case when only an upper bound $\bar{n}$ on n is given.

By Proposition 7, $G^\star$ is unique. We will give an algorithm that maintains a finite set C of graphs that is always guaranteed to contain $G^\star$. In every step our algorithm will rule out at least one member of C, until there is only one left. This graph will then be $G^\star$. Throughout the algorithm, we denote by π_{hist} the label-sequence associated with the walk along which the robot has travelled so far and by v_{hist} the target of the walk. As G is locally oriented, π_{hist} together with the initial starting location of the robot uniquely corresponds to this walk in G. The walk however is not explicitly known to the robot as it neither knows G nor its starting location.

We start by setting C to contain all opfibration prime graphs of size at most n. In every step let G_1 be a graph of minimum size in C and G_2 be a graph of minimum size in $C \backslash \{G_1\}$ (if $C \backslash \{G_1\} = \emptyset$, we are done and set $G^\star = G_1$). We now describe how to discard either G_1 or G_2 from C.

By Lemma 11, there is a label-sequence δ for which $\delta(G_1) = \emptyset$ and $\delta(G_2) \neq \emptyset$ or vice versa. The robot can determine the shortest such label-sequence δ simply by enumerating all possible label-sequences in order of increasing lengths and checking for each in turn whether it has the desired property. Without loss of generality assume $\delta(G_1) = \emptyset$ and $\delta(G_2) \neq \emptyset$. The robot does not explicitly know G nor where in G it was initially located. It thus iterates over all candidate graphs $G' = (V', E')$ of size n and all vertices $v' \in V'$ (again there are only finitely many choices). For every choice of G', let $\Pi(G')$ be the set of all label-sequences associated to walks in G' that have the same length as δ. It is easy to see that there is a label-sequence π of finite length in G' for which $\pi(v_{\text{hist}}(v')) \neq \emptyset$ and which contains all label-sequences in $\Pi(G')$ as a subsequence. The robot follows this label-sequence (because of local orientation, every decision is unique) either until is reaches its end, or until it cannot anymore because there is no arc of the required label emanating from its current vertex.

As we iterate over every choice of G' and v', we are sure to reach G and the robot's initial starting location at some point in the process. We can thus be sure that in the end π_{hist} contains all label-sequences associated to walks in G with the same length as δ as a substring, and of course conversely all substrings of π_{hist} of that length are label-sequences of walks in G. It remains to check whether π_{hist} contains δ as a substring. If yes, we discard G_1 from C and otherwise we discard G_2. We can do this because $\delta(G_1) = \emptyset$ and $\delta(G_2) \neq \emptyset$ and because by Proposition 6, any valid choice for $G^\star$ must have the same set of label-sequences as G. We then continue with new choices for G_1 and G_2. After a finite number of steps C will only contain one graph which is a valid choice for $G^\star$. This concludes the proof.

Observe now that if only an upper bound $\bar{n}$ on n is given, the algorithm can easily be adapted to find $G^\star$ in the same way by iterating over all graphs G' of size *at most* $\bar{n}$ for every pair of graphs G_1, G_2. ◀

We obtain Theorem 1 immediately by applying Theorem 12 to G_{vis}. Note that the results of this section are not restricted to visibility graphs.

5 Identifying the clique $C^\star$

In this section we study structural properties of $G^\star_{\text{vis}} = (V^\star, E^\star)$ which we later use to show Theorem 3.

Let $\varphi : G_{\text{vis}} \to G^\star_{\text{vis}}$ be the opfibration from G_{vis} to $G^\star_{\text{vis}}$. As $G^\star_{\text{vis}}$ is the minimum base of G_{vis}, φ is unique. Every vertex $v^\star$ of $G^\star_{\text{vis}}$ corresponds to a set of vertices of G_{vis}. We write $C_{v^\star} := \varphi^{-1}(v^\star) \subseteq V$ and say $C_{v^\star}$ is the *class* of $v^\star$. For all $v \in \varphi^{-1}(v^\star)$, we set $C_v := C_{v^\star}$. From the definition of opfibrations it follows that every two vertices u, v of the same class C_u have the same degree d and that due to local orientation we have $C_{\text{vis}_u(i)} = C_{\text{vis}_v(i)}$ for all $1 \leq i \leq d$. We may thus write $C_u(i) := C_{\text{vis}_u(i)}$. Finally, we define $\mathcal{B} := \left(C_{v_0}, C_{v_1}, \ldots, C_{v_{n-1}}\right)$ to be the sequence in which the classes appear along the boundary.

As $G^\star_{\text{vis}}$ is opfibration prime, by Proposition 9 every vertex has its unique universal total graph. We use this and define a natural order $\mathcal{O}$ on the vertices of $G^\star_{\text{vis}}$ and thus on the classes of G_{vis}.

▶ **Lemma 13.** *The sequence $\mathcal{B}$ is periodical with period $|V^\star|$ and thus all classes have the same size.*

Proof. The image of the boundary under φ consists of $n/|V^\star|$ copies of a Hamiltonian cycle of $G^\star_{\text{vis}}$. Hence $\mathcal{B}$ is periodical with period $|V^\star|$ and all classes have the size $n/|V^\star|$. ◀

We show that if a vertex from some class is an ear, then every vertex of the class is an ear. Recall that an ear of G_{vis} is a vertex $v_i \in V$ for which v_{i-1} and v_{i+1} see each other. We will need the following property of the shortest curve between two vertices of $\mathcal{P}$.

▶ **Theorem 14** ([11]). *Let $s, t \in V$. There is a unique shortest curve p from s to t that lies in $\mathcal{P}$. This curve is a chain of straight-line segments connected at reflex vertices of $\mathcal{P}$, and the two line segments at any vertex of p form a reflex angle. We say p is the* (euclidean) shortest path *in $\mathcal{P}$ between s and t.*

▶ **Lemma 15.** *Let $|V^\star| > 2$ and $v_x, v_y \in V$ such that $C_{v_x}(2) = C_{v_y}$ and $C_{v_y}(-2) = C_{v_x}$. Then, $C_{v_{x+2}} = C_{v_y}$ and every vertex in $C_{v_{x+1}}$ is an ear.*

Proof. We start by observing that for all $v_i \in V$ with $\text{vis}_{\text{vis}_{v_i}(2)}(-2) = v_i$ we have $\text{vis}_{v_i}(2) = v_{i+2}$ and thus v_{i+1} is an ear. For the sake of contradiction assume $\text{vis}_{\text{vis}_{v_i}(2)}(-2) = v_i$ but

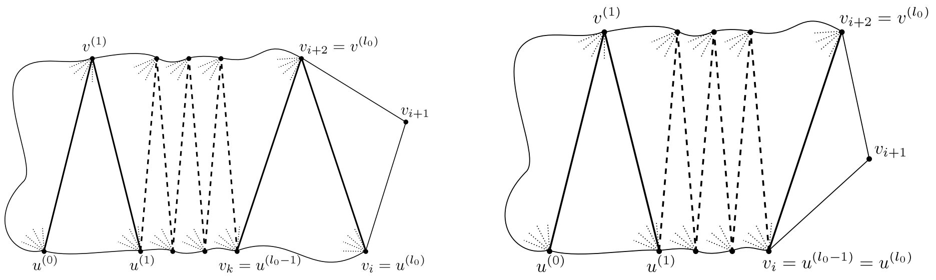


Figure 2 Visualisation of the "zig-zag" sequence Z. As Z does not self-intersect, there is a point l_0 from which on Z's entries do not change anymore. There are two cases how this point is reached: either $u^{(l_0-1)}$ is distinct from $u^{(l_0)}$ (left) or both are the same (right).

$\mathrm{vis}_{v_i}(2) \neq v_{i+2}$. Consider the subpolygon induced by $\mathrm{chain}(v_i, \mathrm{vis}_{v_i}(2))$. This subpolygon has at least four vertices as $\mathrm{vis}_{v_i}(2) \notin \{v_{i+1}, v_{i+2}\}$. In the visibility graph of the subpolygon, v_i and $\mathrm{vis}_{v_i}(2)$ are neighbors on the boundary and both have degree two, which is a contradiction to the fact that every polygon must admit a triangulation. Therefore $\mathrm{vis}_{v_i}(2) = v_{i+2}$ and v_{i+1} is an ear as its neighbors on the boundary see each other.

Because of the above observation, it is sufficient to show that for every $v \in C_{v_x}$ we have $\mathrm{vis}_{\mathrm{vis}_v(2)}(-2) = v$. For the sake of contradiction assume in the following that there is a vertex $u^{(0)} \in C_{v_x}$ with $\mathrm{vis}_{\mathrm{vis}_{u^{(0)}}(2)}(-2) \neq u^{(0)}$.

We define an infinite sequence $Z = \big(u^{(0)}, v^{(1)}, u^{(1)}, v^{(2)}, \ldots\big)$ by $v^{(l)} := \mathrm{vis}_{u^{(l-1)}}(2)$ and $u^{(l)} := \mathrm{vis}_{v^{(l)}}(-2)$ for all $l > 0$. Obviously $u^{(l)} \in C_{v_x}, v^{(l)} \in C_{v_y}$ for all $l \geq 0$. Intuitively, Z is the zig-zag line obtained by alternatingly travelling along the first and the last non-boundary arc in ccw order, starting at $u^{(0)}$. It is immediate to see that for any fixed index $l' \geq 0$ we have $u^{(l)}, v^{(l)} \in \mathrm{chain}(u^{(l')}, v^{(l')})$ for all $l \geq l'$. Hence the part of the boundary in which these vertices lie becomes smaller and smaller and from some index $l_0 \geq 0$ on we have $u^{(l)} = u^{(l_0)}$ and $v^{(l)} = v^{(l_0)}$ for all $l \geq l_0$ (we set l_0 to be the smallest such index). Let $0 \leq i, j < n$ such that $v_i = u^{(l_0)}, v_j = v^{(l_0)}$. We then have $\mathrm{vis}_{v_i}(2) = v_j$ and $\mathrm{vis}_{\mathrm{vis}_{v_i}(2)}(-2) = v_i$. Thus by the above observation, v_{i+1} is an ear and $v_j = v_{i+2}$. As $v_i \in C_{v_x}$ and $v_j \in C_{v_y}$, this implies $C_{v_{x+2}} = C_{v_y}$. It remains to show that every vertex in $C_{v_{x+1}}$ is an ear.

We have to consider two cases. Either $u^{(l_0-1)}$ is distinct from $u^{(l_0)}$ or it is the same vertex (cf. Fig. 2). We assume $u^{(l_0-1)} \neq u^{(l_0)}$ and omit the discussion of the second case which is essentially analogous. Let $0 \leq k < n$ such that $v_k = u^{(l_0-1)}$. As $\mathrm{vis}_{v_k}(2) = v_{i+2}$, we have that v_k does not see any vertex in $\mathrm{chain}(v_{k+2}, v_{i+1})$ (note that this chain is not empty as $v_k \neq v_i$) and thus as $v_{k+1} \in C_{v_{x+1}}$ is in the same class as (the ear) v_{i+1}, the interior angle of the polygon at v_{k+1} is strictly smaller than π. For geometrical reasons (cf. Fig. 3) no vertex in $\mathrm{chain}(v_{i+3}, v_k)$ can see any vertex in $\mathrm{chain}(v_{k+2}, v_{i+1})$. Let $X \subset C_{v_x}$ be the set of vertices of C_{v_x} in $\mathrm{chain}(v_{i+3}, v_k)$ and let $Y \subset C_{v_y}$ be the set of vertices of C_{v_y} in $\mathrm{chain}(v_{i+3}, v_k)$. As $|V^\star| > 2$, $C_{v_x}, C_{v_{x+1}}, C_{v_{x+2}}$ are all different and thus X and Y are disjoint. Note that because $\mathcal{B}$ is periodical with period $|V^\star|$ (Lemma 13) we have $|X| = |Y| + 1$.

We define the (undirected) bipartite graph $B_{xy} = \big(C_{v_x} \cup C_{v_y}, E_{xy}\big)$ with the edge-set $E_{xy} = \big\{\{u, v\} \in C_{v_x} \times C_{v_y} \,|\, (u, v) \in E\big\}$. In B_{xy} all vertices need to have the same degree d as $|C_{v_x}| = \big|C_{v_y}\big|$ and all vertices in either class have the same degree. We have $|X| = |Y| + 1$, we have that vertices in X can only have edges to vertices in $Y \cup \{v_{i+2}\}$ and that vertices in Y can only have edges to vertices in X. For all vertices to have the same degree, v_{i+2} cannot have any edges leading to $C_{v_x} \backslash X$. This is a contradiction to the fact that v_{i+2} sees v_i which is not in $\mathrm{chain}(v_{i+3}, v_k)$ and thus not in X. ◀

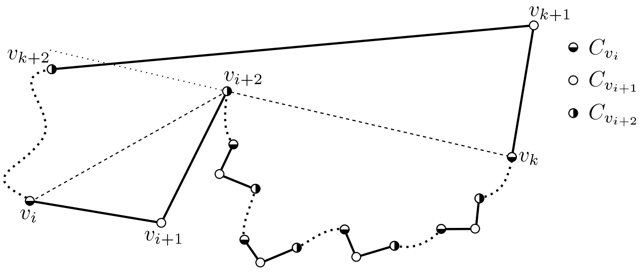


Figure 3 No vertex in chain(v_{i+3}, v_k) can see any vertex in chain(v_{k+2}, v_{i+1}).

We can now consider arbitrary values of $|V^\star|$ and prove Lemma 2.

Proof of Lemma 2. In the following we let $v_i \in V$ be an ear and show that all vertices in C_{v_i} are ears.

First consider the case $|V^\star| > 2$. As $(v_{i-1}, v_{i+1}) \in E$, we have $\text{vis}_{v_{i-1}}(2) = v_{i+1}$ and $\text{vis}_{i+1}(-2) = v_{i-1}$, and thus $C_{v_{i-1}}(2) = C_{v_{i+1}}$ and $C_{v_{i+1}}(-2) = C_{v_{i-1}}$. By Lemma 15 all vertices in C_{v_i} are ears. Now consider the case $|V^\star| = 1$. In that case as v_i is convex, so are all vertices in C_{v_i}, as convexity is encoded in the arc-labeling. As $|V^\star| = 1$, this means that the polygon is convex and thus all vertices are ears.

It remains to consider the case $|V^\star| = 2$. Let $C_{v_j} \neq C_{v_i}$ be the second class in G_{vis}. Again, v_i is convex and thus all vertices in C_{v_i} are. For the sake of contradiction assume that there is a vertex $v_x \in C_{v_i}$ which is not an ear. Then v_{x-1} and v_{x+1} do not see each other, and by Lemma 13, $v_{x-1}, v_{x+1} \in C_{v_j}$. Let p be the shortest path in $\mathcal{P}$ between v_{x-1} and v_{x+1}. By Theorem 14, all vertices on p are reflex. This means that all vertices on p must be from C_{v_j} and thus all vertices of C_{v_j} must be reflex. Moreover, every vertex u in C_{v_j} has two neighbors u', u'' in C_{v_j} such that the angle between (u, u') and (u, u'') is reflex. If we cut off v_i from $\mathcal{P}$, we do not affect this property (every vertex u in C_{v_j} still has two neighbors from C_{v_j} forming a reflex angle) and we thus obtain a new polygon in which all vertices in C_{v_j} are still reflex. We can continue to obtain smaller and smaller subpolygons by selecting ears and cutting them off, maintaining the property that all vertices in C_{v_j} are reflex. Thus, in this process, we never cut off a vertex of C_{v_j}. This is a contradiction, as every polygon has at least one ear and thus the above process has to cut off all vertices eventually. ◀

Lemma 2 allows us to employ the following procedure repeatedly until only one class $C^\star$ remains: In step i, select the class $C^{(i)}$ which is smallest w.r.t. the order $\mathcal{O}$ among all classes of ears. We remove $C^{(i)}$ from the polygon by deleting the corresponding vertex from $G^\star_{\text{vis}}$ and updating the arc labels of its neighborhood accordingly. Removing class $C^{(i)}$ in that way produces a (not necessarily minimum) base graph of the visibility graph of the subpolygon obtained by cutting off all ears in $C^{(i)}$. In the next step we effectively consider this new polygon which again has to have at least one ear, and we are guaranteed to again have at least one class that contains only ears. Note that the above procedure does not require the base graph on which it operates in each step to be *minimum*. We start with the *minimum* base graph $G^\star_{\text{vis}}$ because it is the only base graph of G_{vis} the robot can infer at all.

If we repeat our procedure $|V^\star| - 1$ times, we are left with a single class $C^{(|V^\star|)} = C^\star$ and a sequence $\left(C^{(1)}, C^{(2)}, \ldots, C^{(|V^\star|-1)}\right)$ which is fixed by our order relation $\mathcal{O}$. As $C^\star$ again corresponds to a subpolygon and thus must contain at least one ear, every vertex in $C^\star$ must be an ear. Therefore the corresponding subpolygon is convex and $C^\star$ forms a clique in G_{vis}. This proves Theorem 3.

The existence of a clique gives us a way of computing n from $\bar{n}$ using $G^{\star}_{\text{vis}}$. By Lemma 13 we have $n = |V^{\star}| \cdot |C|$, where C is any class of G_{vis}. If we inspect the number of self-loops of every vertex of $G^{\star}_{\text{vis}}$, we are sure to encounter at least one clique, and thus $|C|$ is equal to the maximum number of self-loops plus one.

By Theorem 1, a robot can determine $G^{\star}$ in finite time. It thus can execute the above procedure and we obtain

▶ **Theorem 16.** *A robot in $\mathcal{P}$ can determine the sequence $\mathcal{C} = \left(C^{(1)}, C^{(2)}, \dots, C^{(|V^{\star}|)}\right)$, where $\mathcal{C}$ is the lexicographically smallest sequence such that for every $1 \le i \le |V^{\star}|$, all vertices in $C^{(i)}$ are ears in the subpolygon obtained by removing all vertices in $\bigcup_{j=1}^{i-1} C^{(j)}$ from $\mathcal{P}$.*

6 Reconstructing the visibility graph

In the following, we assume that the robot has already determined $G^{\star}_{\text{vis}}$ and the sequence $\mathcal{C} = \left(C^{(1)}, C^{(2)}, \dots, C^{(|V^{\star}|-1)}, C^{(|V^{\star}|)}\right)$ from Theorem 16. For all $1 \le i \le |V^{\star}|$ we denote by $G^{(i)}_{\text{vis}} = \left(V^{(i)}, E^{(i)}\right)$ the subgraph of G_{vis} induced by $\bigcup_{j=i}^{|V^{\star}|} C^{(j)}$. By definition of $\mathcal{C}$, $G^{(i)}_{\text{vis}}$ is the visibility graph of a subpolygon of $\mathcal{P}$, and we denote this subpolygon by $\mathcal{P}^{(i)}$. As $C^{(|V^{\star}|)} = C^{\star}$, by Lemma 13 we have that $G^{(|V^{\star}|)}_{\text{vis}}$ is the complete graph on $n/|V^{\star}|$ vertices. Together with the following lemma, this suggests a way of reconstructing $G_{\text{vis}} = G^{(1)}_{\text{vis}}$.

▶ **Lemma 17.** *Let $1 \le i < |V^{\star}|$. It is possible to determine $G^{(i)}_{\text{vis}}$ from $G^{(i+1)}_{\text{vis}}$.*

Proof. The set of vertices $V^{(i)}$ of $G^{(i)}_{\text{vis}}$ is given by $V^{(i)} = C^{(i)} \cup V^{(i+1)}$. It remains to show how to construct $E^{(i)}$. Let A be the set of arcs in G_{vis} between vertices of $C^{(i)}$ and $V^{(i+1)}$, and B be the set of arcs between vertices of $C^{(i)}$. We will first show how to construct A using the information contained in $G^{(i+1)}_{\text{vis}}$ and $G^{\star}_{\text{vis}}$. After having determined A, we can apply the same approach in order to obtain B. This completes the proof as $E^{(i)} = E^{(i+1)} \cup A \cup B$.

Note that every arc in G_{vis} has a counterpart of opposite orientation. In order to construct A it is thus sufficient to consider $e \in V^{(i+1)} \times C^{(i)}$ and show how to decide whether $e \in A$ or $e \notin A$. Deciding which elements of $C^{(i)} \times V^{(i+1)}$ are in A is then immediate. Equivalently, we can consider $v_j \in V^{(i+1)}$ with degree d in $G^{(i)}_{\text{vis}}$ and $1 \le k \le d$ such that $\text{vis}_{v_j}(k) \in C^{(i)}$, and show how to "identify" $\text{vis}_{v_j}(k)$, i.e. how to find x with $v_x = \text{vis}_{v_j}(k)$. If $k = 1$, we have $x = j + 1$ and if $k = d$, we have $x = j - 1$ because v_j sees its two neighbors on the boundary. Now assume $1 < k < d$. We will show that $v_y := \text{vis}_{v_j}(k-1) \notin C^{(i)}$. For the sake of contradiction assume that $v_y \in C^{(i)}$. In $\mathcal{P}^{(i)}$ all vertices of $C^{(i)}$ are ears and thus convex. By Lemma 13 and $i < |V^{\star}|$, there is more than one class and thus there is a vertex $v_z \in \text{chain}(v_{y+1}, v_{x-1})$ which is not visible to v_j. The shortest path in $\mathcal{P}$ from v_j to v_z must visit v_x or v_y, which is a contradiction to both vertices being convex (Theorem 14). We can deduce that $v_y \notin C^{(i)}$ and thus $(v_j, v_y) \in E^{(i+1)}$ is part of $G^{(i+1)}_{\text{vis}}$ and has already been identified, i.e. the index y is known. Because of Lemma 13, it is sufficient to know how many vertices of $C^{(i)}$ are in $\text{chain}(v_{y+1}, v_{x-1})$ in order to find x itself. From the labeling of $G^{\star}_{\text{vis}}$ we can deduce how many vertices of $C^{(i)}$ are in $\text{chain}_{v_y}(v_{y+1}, v_{x-1})$ (recall that $\text{chain}_{v_y}(v_{y+1}, v_{x-1})$ only contains vertices visible to v_y): As v_x is convex and thus cannot lie on a shortest path in $\mathcal{P}$ from v_j to another vertex of $C^{(i)}$, the first arc in ccw order from v_y to a vertex of $C^{(i)}$ that forms a convex angle with (v_y, v_j) must be (v_y, v_x) as the target of the arc must be visible to v_j. It is thus sufficient to count the number of arcs from v_y to vertices of $C^{(i)}$ before (v_y, v_x) in ccw order. We say the corresponding vertices are *hidden from v_j by v_y*. We still need a way to find the number of vertices of $C^{(i)}$ in $\text{chain}(v_{y+1}, v_{x-1})$ that are *not* visible to v_y. We can find this number by repeating our counting method recursively.

For every vertex $v_l \in \text{chain}_{v_y}(v_{y+1}, v_{x-1}) \setminus C^{(i)}$ (in $G_{\text{vis}}^{(i)}$), we count all vertices of $C^{(i)}$ hidden from v_y by v_l. As the vertices in $C^{(i)}$ are convex, they cannot hide any vertices from v_y. The sum of all these counts finally gives the number of vertices of $C^{(i)}$ in $\text{chain}(v_{y+1}, v_{x-1})$. Together with Lemma 13 this number immediately yields the index x. The recursive counting method described above was first introduced in a similar setting where robots are allowed to retrace their movements [3]. Refer to [3] for a detailed proof of its correctness.

Using the fact that the arcs in A have already been identified, we can apply the exact same approach to construct B. ◀

Theorem 5 follows directly from Theorem 16, Lemma 17 and the fact that $G^{(|V^\star|)}$ is the complete graph on $n/|V^\star|$ vertices.

References

1 H. Ando, Y. Oasa, I. Suzuki, and M. Yamashita. Distributed memoryless point convergence algorithm for mobile robots with limited visibility. *IEEE Transactions on Robotics and Automation*, 15(5):818–828, 1999.

2 D. Angluin. Local and global properties in networks of processors. In *Proceedings of the twelfth annual ACM symposium on Theory of computing*, pages 82–93, 1980.

3 D. Bilò, Y. Disser, M. Mihalák, S. Suri, E. Vicari, and P. Widmayer. Reconstructing visibility graphs with simple robots. In *Proceedings of the 16th International Colloquium on Structural Information and Communication Complexity*, pages 87–99, 2009.

4 P. Boldi and S. Vigna. Fibrations of graphs. *Discrete Mathematics*, 243(1–3):21–66, 2002.

5 J. Brunner, M. Mihalák, S. Suri, E. Vicari, and P. Widmayer. Simple robots in polygonal environments: A hierarchy. In *Proceedings of the Fourth International Workshop on Algorithmic Aspects of Wireless Sensor Networks*, pages 111–124, 2008.

6 J. Chalopin, S. Das, Y. Disser, M. Mihalák, and P. Widmayer. How simple robots benefit from looking back. In *Proceedings of the 7th International Conference on Algorithms and Complexity*, pages 229–239, 2010.

7 J. Chalopin, E. Godard, Y. Métivier, and R. Ossamy. Mobile agent algorithms versus message passing algorithms. In *Proceedings of the 10th International Conference on the Principles of Distributed Systems*, pages 187–201, 2006.

8 R. Cohen and D. Peleg. Convergence of autonomous mobile robots with inaccurate sensors and movements. *SIAM Journal on Computing*, 38(1):276–302, 2008.

9 Y. Disser, M. Mihalák, and P. Widmayer. Reconstructing a simple polygon from its angles. In *Proceedings of the 12th Scandinavian Symposium and Workshops on Algorithm Theory*, pages 13–24, 2010.

10 A. Ganguli, J. Cortés, and F. Bullo. Distributed deployment of asynchronous guards in art galleries. In *Proceedings of the 2006 American Control Conference*, pages 1416–1421, 2006.

11 D.-T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

12 S. Suri, E. Vicari, and P. Widmayer. Simple robots with minimal sensing: From local visibility to global geometry. *International Journal of Robotics Research*, 27(9):1055–1067, 2008.

13 M. Yamashita and T. Kameda. Computing on anonymous networks: Part I – characterizing the solvable cases. *IEEE Transactions on Parallel and Distributed Systems*, 7(1):69–89, 1996.

14 A. Yershova, B. Tovar, R. Ghrist, and S. M. LaValle. Bitbots: Simple robots solving complex tasks. In *Proceedings of the 20th national conference on Artificial intelligence*, volume 3, pages 1336–1341, 2005.

Cross-Composition: A New Technique for Kernelization Lower Bounds*

Hans L. Bodlaender[1], Bart M. P. Jansen[1], and Stefan Kratsch[1]

1 **Department of Information and Computing Sciences, Utrecht University**
P.O. Box 80.089, 3508 TB, Utrecht, The Netherlands
{hansb,bart,kratsch}@cs.uu.nl

Abstract

We introduce a new technique for proving kernelization lower bounds, called *cross-composition*. A classical problem L cross-composes into a parameterized problem Q if an instance of Q with polynomially bounded parameter value can express the logical OR of a sequence of instances of L. Building on work by Bodlaender et al. (ICALP 2008) and using a result by Fortnow and Santhanam (STOC 2008) we show that if an NP-hard problem cross-composes into a parameterized problem Q then Q does not admit a polynomial kernel unless the polynomial hierarchy collapses.

Our technique generalizes and strengthens the recent techniques of using OR-composition algorithms and of transferring the lower bounds via polynomial parameter transformations. We show its applicability by proving kernelization lower bounds for a number of important graphs problems with structural (non-standard) parameterizations, e.g., Chromatic Number, Clique, and Weighted Feedback Vertex Set do not admit polynomial kernels with respect to the vertex cover number of the input graphs unless the polynomial hierarchy collapses, contrasting the fact that these problems are trivially fixed-parameter tractable for this parameter. We have similar lower bounds for Feedback Vertex Set.

1998 ACM Subject Classification F.2.2

Keywords and phrases kernelization, lower bounds, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.165

1 Introduction

Preprocessing and data reduction are important and widely applied concepts for speeding up polynomial-time algorithms or for making computation feasible at all in the case of hard problems that are not believed to have efficient algorithms. Kernelization is a way of formalizing data reduction, which allows for a formal analysis of the (im)possibility of data reduction and preprocessing. It originated as a technique to obtain fixed-parameter tractable algorithms for hard (parameterized) problems, and has evolved into its own topic of research (see [19, 2] for recent surveys). A *parameterized problem* [14, 16] is a language $Q \subseteq \Sigma^* \times \mathbb{N}$, the second component is called the *parameter*. A *kernelization* algorithm (*kernel*) transforms an instance (x, k) in polynomial time into an equivalent instance (x', k') such that $|x'|, k' \leq f(k)$ for some computable function f, which is the *size* of the kernel.

From a practical perspective we are particularly interested in cases where $f \in k^{O(1)}$, so-called *polynomial kernels*. Success stories of kernelization include the $O(k^2)$ kernel for k-Vertex Cover containing at most $2k$ vertices [11] and the meta-theorems for kernelization

* This work was supported by the Netherlands Organisation for Scientific Research (NWO), project "KERNELS: Combinatorial Analysis of Data Reduction".

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 165–176
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

of problems on planar graphs [4], among many others (cf. also [22]). Although researchers have looked for polynomial kernels for elusive problems such as k-Path for many years, it was only recently that techniques were introduced which make it possible to prove (under some complexity-theoretic assumption) that a parameterized problem in FPT does not admit a polynomial kernel. Bodlaender et al. [3] introduced the concept of a or-*composition* algorithm as a tool to give super-polynomial lower bounds on kernel sizes. Consider some set S, and let $\textsc{or}(S)$ denote the set such that for any sequence $x^* := (x_1, \ldots, x_t)$ of instances of S we have $x^* \in \textsc{or}(S) \Leftrightarrow \bigvee_{i=1}^{t} x_i \in S$; then we could say that the language $\textsc{or}(S)$ expresses the or of instances of S. The approach taken in the original paper by Bodlaender et al. [3] uses a theorem by Fortnow and Santhanam [17] to show that if there is a polynomial-time or-composition algorithm that maps any sequence of instances $(x_1, k), (x_2, k), \ldots, (x_t, k)$ of some parameterized problem Q which all share the same parameter value to an instance (x^*, k^*) of Q which acts as the or of the inputs and $k^* \in k^{O(1)}$, then Q does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly. This machinery made it possible to prove e.g. that k-Path and the Clique problem parameterized by the treewidth of the graph do not admit polynomial kernels unless NP $\subseteq$ coNP/poly[1]. The latter is deemed unlikely since it is known to imply a collapse of the polynomial hierarchy to its third level [24] (and further [8]).

It did not take long before the techniques of Bodlaender et al. were combined with the notion of a *polynomial parameter transformation* to also prove lower bounds for problems for which no direct or-composition algorithm could be found. This idea was used implicitly by Fernau et al. [15] to show that k-Leaf Out-Branching does not admit a polynomial kernel, and was formalized in a paper by Bodlaender et al. [7]: they showed that if there is a polynomial-time transformation from P to Q which incurs only a polynomial blow-up in the parameter size, then if P does not admit a polynomial kernel then Q does not admit one either. These polynomial parameter transformations were used extensively by Dom et al. [13] who proved kernelization lower bounds for a multitude of important parameterized problems such as Small Universe Hitting Set and Small Universe Set Cover. Dell and van Melkebeek [12] were able to extend the techniques of Fortnow and Santhanam to prove, e.g., that Vertex Cover does not admit a kernel of size $O(k^{2-\epsilon})$ for any $\epsilon > 0$.

Our results. We introduce a new technique to prove kernelization lower-bounds, which we call *cross-composition.* This technique generalizes and strengthens the earlier methods of or-composition [3] and polynomial-parameter transformations [7], and puts the two existing methods of showing kernelization lower bounds in a common perspective. Whereas the existing notion of or-composition works by composing multiple instances of a *parameterized* problem Q into a single instance of Q with a bounded parameter value, for our new technique it is sufficient to compose the or of any *classical* NP-hard problem into an instance of the parameterized problem Q for which we want to prove a lower-bound. The term *cross* in the name stems from this fact: the source- and target problem of the composition need no longer be the same. Since the input to a cross-composition algorithm is a list of *classical* instances instead of parameterized instances, the inputs do not have a parameter in which the output parameter of the composition must be bounded; instead we require that the size of the output parameter is polynomially bounded in the size of the largest input instance. In addition we show that the output parameter may depend polynomially on the logarithm of the number of input instances, which often simplifies the constructions and proofs. We also introduce the concept of a *polynomial equivalence relation* to remove the need for padding

[1] In the remainder of this introduction we assume that NP $\not\subseteq$ coNP/poly when stating kernelization lower bounds.

Problem name	Parameter	Kernel size	
CLIQUE	vertex cover	not polynomial	[Section 4.1]
CHROMATIC NUMBER	vertex cover	not polynomial	[Section 4.2]
FEEDBACK VERTEX SET	dist. from cluster	not polynomial	[Section 4.3]
FEEDBACK VERTEX SET	dist. from co-cluster	not polynomial	[Section 4.3]
WEIGHTED FVS	vertex cover	not polynomial	[Section 4.3]

Table 1 An overview of the kernelization lower bounds obtained in this paper; all listed problems are fixed-parameter tractable with respect to this parameterization. Section 4 describes the parameterized problems in more detail.

arguments which were frequently required for OR-compositions.

To show the power of cross-composition we give kernelization lower bounds for *structural* parameterizations of several important graph problems. Since many combinatorial problems are easy on graphs of bounded treewidth [6], and since the treewidth of a graph is bounded by the vertex cover number, it is often thought that almost all problems become tractable when parameterized by the vertex cover number of the graph. We show that this is not the case for kernelization: CLIQUE, CHROMATIC NUMBER and WEIGHTED FEEDBACK VERTEX SET do not admit polynomial kernels parameterized by the vertex cover number of the graph. In the case of CLIQUE it was already known [3] that the problem does not admit a polynomial kernel parameterized by the treewidth of the graph; since the vertex cover number is at least as large as the treewidth we prove a stronger result. For the unweighted FEEDBACK VERTEX SET problem, which admits a polynomial kernel parameterized by the target size of the feedback set [23], we show that there is no polynomial kernel for the parameterization by deletion distance to cluster graphs or co-cluster graphs.

Organization. The paper is organized as follows. We first give some preliminary definitions. Section 3 gives the formal definition of cross-composition, and proves that cross-compositions allow us to give kernelization lower bounds. In Section 4 we apply the new technique to obtain kernelization lower bounds for various problems.

2 Preliminaries

In this work we only consider undirected, finite, simple graphs. Let G be a graph and denote its vertex set by $V(G)$ and the edge set by $E(G)$. We use $\chi(G)$ to denote the chromatic number of G. If $V' \subseteq V(G)$ then $G[V']$ denotes the subgraph of G induced by V'. A graph is a *cluster graph* if every connected component is a clique. A graph is a *co-cluster* graph if it is the edge-complement of a cluster graph. Throughout this work we use Σ to denote a finite alphabet, but note that multiple occurrences of Σ may refer to different alphabets. For positive integers n we define $[n] := \{1, \ldots, n\}$. The satisfiability problem for boolean formulae is referred to as SAT. Several proofs have been deferred to the full version [5] of this paper due to space restrictions. For completeness we give the following core definitions of parameterized complexity [3, 14].

▶ **Definition 1.** A *parameterized problem* is a language $Q \subseteq \Sigma^* \times \mathbb{N}$, and is contained in the class (strongly uniform) FPT (for Fixed-Parameter Tractable) if there is an algorithm that decides whether $(x, k) \in Q$ in $f(k)|x|^{O(1)}$ time for some computable function f.

▶ **Definition 2.** A *kernelization* algorithm [19, 2], or in short, a *kernel* for a parameterized problem $Q \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that given $(x, k) \in \Sigma^* \times \mathbb{N}$ outputs in $p(|x| + k)$ time a pair $(x', k') \in \Sigma^* \times \mathbb{N}$ such that:

- $(x, k) \in Q \Leftrightarrow (x', k') \in Q$,
- $|x'|, k' \leq f(k)$,

where f is a computable function, and p a polynomial. Any function f as above is referred to as the size of the kernel; if f is a polynomial then we have a *polynomial kernel.*

3 Cross-Composition

3.1 The Definition

In this section we define the concept of cross-composition and give all the terminology needed to apply the technique.

▶ **Definition 3** (Polynomial equivalence relation)**.** An equivalence relation $\mathcal{R}$ on Σ^* is called a *polynomial equivalence relation* if the following two conditions hold:

1. There is an algorithm that given two strings $x, y \in \Sigma^*$ decides whether x and y belong to the same equivalence class in $(|x| + |y|)^{O(1)}$ time.
2. For any finite set $S \subseteq \Sigma^*$ the equivalence relation $\mathcal{R}$ partitions the elements of S into at most $(\max_{x \in S} |x|)^{O(1)}$ classes.

▶ **Definition 4** (Cross-composition)**.** Let $L \subseteq \Sigma^*$ be a set and let $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. We say that L *cross-composes* into Q if there is a polynomial equivalence relation $\mathcal{R}$ and an algorithm which, given t strings $x_1, x_2, \ldots, x_t$ belonging to the same equivalence class of $\mathcal{R}$, computes an instance $(x^*, k^*) \in \Sigma^* \times \mathbb{N}$ in time polynomial in $\sum_{i=1}^{t} |x_i|$ such that:

1. $(x^*, k^*) \in Q \Leftrightarrow x_i \in L$ for some $1 \leq i \leq t$,
2. k^* is bounded by a polynomial in $\max_{i=1}^{t} |x_i| + \log t$.

3.2 How Cross-compositions Yield Lower Bounds

The purpose of this section is to prove that cross-compositions yield kernelization lower bounds. To give this proof we need some concepts from earlier work [3, 17, 12].

▶ **Definition 5** ([17])**.** A *weak distillation of* SAT into a set $L \subseteq \Sigma^*$ is an algorithm that:

- receives as input a sequence $(x_1, \ldots, x_t)$ of instances of SAT,
- uses time polynomial in $\sum_{i=1}^{t} |x_i|$,
- and outputs a string $y \in \Sigma^*$ with
  1. $y \in L \Leftrightarrow x_i \in$ SAT for some $1 \leq i \leq t$,
  2. $|y|$ is bounded by a polynomial in $\max_{i=1}^{t} |x_i|$.

▶ **Theorem 6** (Theorem 1.2 [17])**.** *If there is a weak distillation of* SAT *into any set $L \subseteq \Sigma^*$ then NP $\subseteq$ coNP/poly and the polynomial-time hierarchy collapses to the third level (PH $= \Sigma_3^p$).*

▶ **Definition 7** ([12])**.** The OR *of a language $L \subseteq \Sigma^*$* is the set OR(L) that consists of all tuples $(x_1, \ldots, x_t)$ for which there is an index $1 \leq i \leq t$ with $x_i \in L$.

▶ **Definition 8** ([3])**.** We associate an instance (x, k) of a parameterized problem with the *unparameterized instance* formed by the string $x\#1^k$, where $\#$ denotes a new character that we add to the alphabet and 1 is an arbitrary letter in Σ. The *unparameterized version* of a parameterized problem Q is the language $\tilde{Q} = \{x\#1^k \mid (x, k) \in Q\}$.

▶ **Theorem 9.** *Let $L \subseteq \Sigma^*$ be a set which is NP-hard under Karp reductions. If L cross-composes into the parameterized problem Q and Q has a polynomial kernel then there is a weak distillation of* SAT *into* OR$(\tilde{Q})$ *and NP $\subseteq$ coNP/poly.*

Proof. The proof is by construction and generalizes the concepts of Bodlaender et al. [3]. Assuming the conditions in the statement of the theorem hold, we show how to build an algorithm which distills SAT into OR$(\tilde{Q})$. By the definition of cross-composition there is a polynomial equivalence relation $\mathcal{R}$ and an algorithm C which composes L-instances belonging to the same class of $\mathcal{R}$ into a Q-instance.

The input to the distillation algorithm consists of a sequence $(x_1, \ldots, x_t)$ of instances of SAT, which we may assume are elements of Σ^*. Define $m := \max_{j=1}^{t} |x_j|$. If $t > (|\Sigma|+1)^m$ then there must be duplicate inputs, since the number of distinct inputs of length $m' \leq m$ is $|\Sigma|^{m'}$. By discarding duplicates we may therefore assume that $t \leq (|\Sigma|+1)^m$, i.e., $\log t \in O(m)$. By the assumption that L is NP-hard under Karp reductions, there is a polynomial-time reduction from SAT to L. We use this reduction to transform each SAT instance x_i for $1 \leq i \leq t$ into an equivalent L-instance y_i. Since the transformation takes polynomial time, it cannot increase the size of an instance by more than a polynomial factor and therefore $|y_i|$ is polynomial in m for all i.

The algorithm now pairwise compares instances using the polynomial-time equivalence test of $\mathcal{R}$ (whose existence is guaranteed by Definition 3) to partition the L-instances $(y_1, \ldots, y_t)$ into partite sets $Y_1, \ldots, Y_r$ such that all instances from the same partite set are equivalent under $\mathcal{R}$. The properties of a polynomial equivalence relation guarantee that r is polynomial in m and that this partitioning step takes polynomial time in the total input size.

We now use the cross-composition algorithm C on each of the partite sets $Y_1, \ldots, Y_r$, which is possible since all instances from the same set are equivalent under $\mathcal{R}$. Let (z_i, k_i) be the result of applying C to a sequence containing the contents of the set Y_i, for $1 \leq i \leq r$. From the definition of cross-composition and using $\log t \in O(m)$ it follows that each k_i is polynomial in m, and that the computation of these parameterized instances takes polynomial time in the total input size. From Definition 4 it follows that (z_i, k_i) is a YES instance of Q if and only if one of the instances in Y_i is a YES instance of L, which in turn happens if and only if one of the inputs x_i is a YES instance of SAT.

Let K be a polynomial kernelization algorithm for Q, whose existence we assumed in the statement of the theorem. We apply K to the instance (z_i, k_i) to obtain an equivalent instance (z'_i, k'_i) of Q for each $1 \leq i \leq r$. Since K is a polynomial kernelization we know that these transformations can be carried out in polynomial time and that $|z'_i|, k'_i \leq k_i^{O(1)}$. Since k_i is polynomial in m it follows that $|z'_i|$ and k'_i are also polynomial in m for $1 \leq i \leq r$.

As the next step we convert each parameterized instance (z'_i, k'_i) to the unparameterized variant $\tilde{z}_i := z'_i\#1^{k'_i}$. Since the values of the parameters are polynomial in m this transformation takes polynomial time, and afterwards we find that $|\tilde{z}_i|$ is polynomial in m for each $1 \leq i \leq r$.

The last stage of the algorithm simply combines all unparameterized variants into one tuple $x^* := (\tilde{z}_1, \tilde{z}_2, \ldots, \tilde{z}_r)$. Since the size of each component is polynomial in m, and since the number of components r is polynomial in m, we have that $|x^*|$ is polynomial in m. The tuple x^* forms an instance of OR$(\tilde{Q})$, and by the definition of OR$(\tilde{Q})$ we know that $x^* \in$ OR$(\tilde{Q})$ if and only if some element of the tuple is contained in $\tilde{Q}$. By tracing back the series of equivalences we therefore find that $x^* \in$ OR$(\tilde{Q})$ if and only if some input x_i is a YES-instance of SAT. Since we can construct x^* in polynomial time and $|x^*|$ is polynomial in m, we have constructed a weak distillation of SAT into OR$(\tilde{Q})$. By Theorem 6 this implies NP $\subseteq$ coNP/poly and proves the theorem. ◀

▶ **Corollary 10.** *If some set L is NP-hard under Karp reductions and L cross-composes into the parameterized problem Q then there is no polynomial kernel for Q unless $NP \subseteq coNP/poly$.*

A simple extension of Theorem 9 shows that cross-compositions also exclude the possibility of compression into a small instance of a different parameterized problem, a notion sometimes referred to as bikernelization [20, 21]. If an NP-hard set cross-composes into a parameterized problem Q, then unless NP $\subseteq$ coNP/poly there is no polynomial-time algorithm that maps an instance (x, k) of Q to an equivalent instance (x', k') of *any* parameterized problem P with $|x'|, k' \leq k^{O(1)}$.

4 Results Based on Cross-Composition

In this section we apply the cross-composition technique to give kernelization lower bounds. We consider the problems Feedback Vertex Set, Chromatic Number and Clique under various parameterizations. The first parameter we consider is the vertex cover number of a graph G, i.e. the cardinality of a smallest set of vertices $Z \subseteq V(G)$ such that all edges of G have at least one endpoint in Z. We show that Clique, Chromatic Number and Weighted Feedback Vertex Set do *not* admit polynomial kernels parameterized by the size of a vertex cover unless NP $\subseteq$ coNP/poly.

We could also define the vertex cover number as the minimum number of vertex deletions needed to reduce a graph to an edgeless graph; hence the vertex cover number measures how far a graph is from being edgeless. Following the initiative of Cai [9] we may similarly define the deletion distance of a graph G to a (co-)cluster graph as the minimum number of vertices that have to be deleted from G to turn it into a (co-)cluster graph. Since (co-)cluster graphs have a very restricted structure, one would expect that a parameterization by (co-) cluster deletion distance leads to fixed-parameter tractability; indeed this is the case for many problems, since graphs of bounded (co-)cluster deletion distance also have bounded cliquewidth [1]. For the Feedback Vertex Set problem, which admits a polynomial kernel parameterized by the target size and hence by the vertex cover number, we show that the parameterizations by cluster deletion or co-cluster deletion distance do not admit polynomial kernels.

In Table 2 we give the known results for our subject problems with respect to the standard parameterization, which refers to the solution size. Since the problems we study are very well-known, we do not give a full definition for each one. Instead we give an educative example of how the parameter is reflected in an instance.

Chromatic Number parameterized by the size of a vertex cover
Instance: A graph G, a vertex cover $Z \subseteq V(G)$, and a positive integer ℓ.
Parameter: The size $k := |Z|$ of the vertex cover.
Question: Is $\chi(G) \leq \ell$, i.e., can G be colored with at most ℓ colors?

For technical reasons we supply a vertex cover in the input of the problem, to ensure that well-formed instances can be recognized in polynomial time. The parameter to the problem claims a bound on the vertex cover number of the graph, and using the set Z we may verify this bound. For Feedback Vertex Set parameterized by deletion distance to cluster graphs or co-cluster graphs, we also supply the deletion set in the input. These versions of the problem are certainly no harder to kernelize than the versions where a deletion set or vertex cover is not given.

Problem name	Parameter	Param. complexity	Kernel size
CLIQUE	clique	W[1]-hard [14]	W[1]-hard [14]
FEEDBACK VERTEX SET	feedback vertex set	FPT [10]	$4k^2$ vertices [23]
CHROMATIC NUMBER	chromatic number	NP-h for $k \in O(1)$	NP-h for $k \in O(1)$

Table 2 Parameterized complexity and kernel size for some of the problems considered in this paper, with respect to the standard parameterization (i.e., target size).

4.1 Clique parameterized by Vertex Cover

An instance of the NP-complete CLIQUE problem [18, GT19] is a tuple (G, ℓ) and asks whether the graph G contains a clique on ℓ vertices. We use this problem for our first kernelization lower bound.

▶ **Theorem 11.** CLIQUE *parameterized by the size of a vertex cover does not admit a polynomial kernel unless NP* $\subseteq$ *coNP/poly.*

Proof. We prove the theorem by showing that CLIQUE cross-composes into CLIQUE parameterized by vertex cover; by Corollary 10 this is sufficient to establish the claim. We define a polynomial equivalence relation $\mathcal{R}$ such that all bitstrings which do not encode a valid instance of CLIQUE are equivalent, and two well-formed instances (G_1, ℓ_1) and (G_2, ℓ_2) are equivalent if and only if they satisfy $|V(G_1)| = |V(G_2)|$ and $\ell_1 = \ell_2$. From this definition it follows that any set of well-formed instances on at most n vertices each is partitioned into $O(n^2)$ equivalence classes. Since all malformed instances are in one class, this proves that $\mathcal{R}$ is indeed a polynomial equivalence relation.

We now give a cross-composition algorithm which composes t input instances $x_1, \ldots, x_t$ which are equivalent under $\mathcal{R}$ into a single instance of CLIQUE parameterized by vertex cover. If the input instances are malformed or the size of the clique that is asked for exceeds the number of vertices in the graph, then we may output a single constant-size NO instance; hence in the remainder we may assume that all inputs are well-formed and encode structures $(G_1, \ell), \ldots, (G_t, \ell)$ such that $|V(G_i)| = n$ for all $i \in [t]$ and all instances agree on the value of ℓ, which is at most n. We construct a single instance (G', Z', ℓ', k') of CLIQUE parameterized by vertex cover, which consists of a graph G' with vertex cover $Z' \subseteq V(G')$ of size k' and an integer ℓ'.

Let the vertices in each G_i be numbered arbitrarily from 1 to n. We construct the graph G' as follows (see also Figure 1):

1. Create ℓn vertices $v_{i,j}$ with $i \in [\ell]$ and $j \in [n]$. Connect two vertices $v_{i,j}$ and $v_{i',j'}$ if $i \neq i'$ and $j \neq j'$. Let C denote the set of these vertices. It is crucial that any clique in G' can only contain one vertex $v_{i,\cdot}$ or $v_{\cdot,j}$ for each choice of $i \in [\ell]$ respectively $j \in [n]$. Thus any clique contains at most ℓ vertices from C.
2. For each pair $1 \leq p < q \leq n$ of distinct vertices from $[n]$ (i.e., vertices of graphs G_i), create three vertices: $w_{p,q}$, $w_{p,\hat{q}}$, and $w_{\hat{p},q}$ and make them adjacent to C as follows:
   - **a.** $w_{p,q}$ is adjacent to all vertices from C,
   - **b.** $w_{p,\hat{q}}$ is adjacent to all vertices from C except for $v_{\cdot,j}$ with $j = q$, and
   - **c.** $w_{\hat{p},q}$ is adjacent to all vertices from C except for $v_{\cdot,j}$ with $j = p$.

   Furthermore we add all edges between vertices $w_{\cdot,\cdot}$ that correspond to distinct pairs from $[n]$. Let D denote these $3\binom{n}{2}$ vertices. Any clique can contain at most one $w_{\cdot,\cdot}$ vertex for each pair from $[n]$.

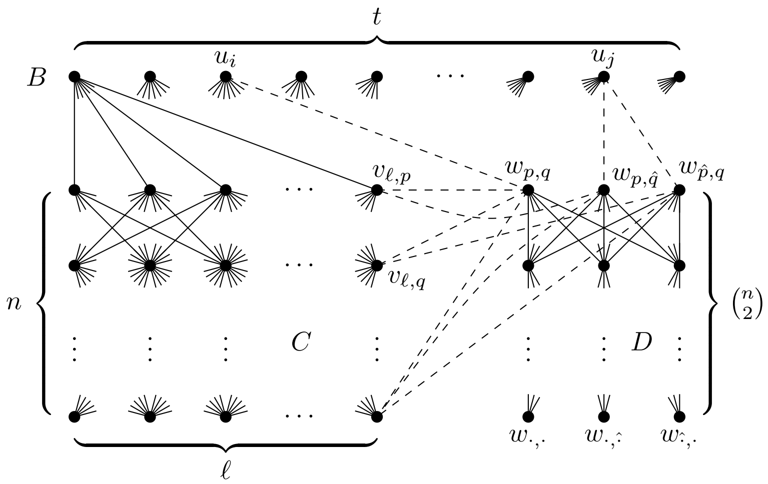


Figure 1 A sketch of the construction used in the proof of Theorem 11. The dashed edges show in an examplary way how vertices $w_{p,q}$, $w_{p,\hat{q}}$, and $w_{\hat{p},q}$ are connected to vertices of B and C, e.g., $\{p,q\}$ is an edge of G_i but not of G_j.

3. For each instance x_i with graph G_i make a new vertex u_i and connect it to all vertices in C. The adjacency to D is as follows:

   a. Make u_i adjacent to $w_{p,q}$ if $\{p,q\}$ is an edge in G_i.
   b. Otherwise make u_i adjacent to $w_{p,\hat{q}}$ and $w_{\hat{p},q}$.

   Let B denote this set of t vertices.

We define $\ell' := \ell + 1 + \binom{n}{2}$. Furthermore, we let $Z' := C \cup D$ which is easily verified to be a vertex cover for G' of size $k' := |Z'| = \ell n + 3\binom{n}{2}$. The value k' is the parameter to the problem, which is polynomial in n and hence in the size of the largest input instance. The cross-composition outputs the instance $x' := (G', Z', \ell', k')$. It is easy to see that our construction of G' can performed in polynomial time. Let us now argue that x' is YES if and only if at least one of the instances x_i is YES.

($\Leftarrow$) First we will assume that some x_{i^*} is YES, i.e., that G_{i^*} contains a clique on at least ℓ vertices. Let $S \subseteq [n]$ denote a clique of size exactly ℓ in G_{i^*}. We will construct a set S' of size $\ell' = \ell + 1 + \binom{n}{2}$ and show that it is a clique in G':

1. We add the vertex u_{i^*} to S'.
2. Let $S = \{p_1, \ldots, p_\ell\} \subseteq [n]$. For each p_j in S we add the vertex v_{j,p_j} to S'. By Step 1 all these vertices are pairwise adjacent, and by Step 3 they are adjacent to u_{i^*}.
3. For each pair $1 \leq p < q \leq n$ there are two cases:

   a. If $\{p,q\}$ is an edge of G_{i^*} then the vertex u_{i^*} is adjacent to $w_{p,q}$ in G' (by Step 3) and $w_{p,q}$ is adjacent to all vertices of C (by Step 2). We add $w_{p,q}$ to S'.
   b. Otherwise the vertex u_{i^*} is adjacent to both $w_{p,\hat{q}}$ and $w_{\hat{p},q}$. Since the clique S cannot contain both p and q when $\{p,q\}$ is a non-edge we are able to add $w_{p,\hat{q}}$ respectively $w_{\hat{p},q}$ to S'; recall that, e.g., $w_{p,\hat{q}}$ is adjacent to all vertices of C except those corresponding to q.

   In both cases we add one $w_{\cdot,\cdot}$-vertex to S', each corresponding to a different pair p,q; all these vertices are pairwise adjacent by Step 2.

We have identified the clique S' in G' of size $\ell' = \ell + 1 + \binom{n}{2}$, proving that x' is a YES-instance.

($\Rightarrow$) Now assume that x' is a YES-instance and let S' be a clique of size $\ell + 1 + \binom{n}{2}$ in G'. Since S' contains at most ℓ vertices from C (i.e., one $v_{i,\cdot}$ for each $i \in [\ell]$) and at most $\binom{n}{2}$ vertices from D it must contain at least one vertex from B, say $u_{i^*} \in B$. Since B is an independent set the set S' must contain exactly ℓ vertices from C and exactly $\binom{n}{2}$ vertices from D. Let $S = \{j \in [n] \mid v_{i,j} \in S' \text{ for some } i \in [\ell]\}$. The set S has size ℓ since S' contains at most one vertex $v_{\cdot,j}$ for each $j \in [n]$. We will now argue that S is a clique in G_{i^*}. Let $p, q \in S$. The clique S' must contain a $w_{\cdot,\cdot}$-vertex corresponding to $\{p,q\}$ and it must contain vertices $v_{i,p}$ and $v_{i',q}$ for some $i, i' \in [\ell]$. Therefore it must contain $w_{p,q}$ since $w_{p,\hat{q}}$ has no edges to vertices $v_{\cdot,q}$ and $w_{\hat{p},q}$ has no edges to $v_{\cdot,p}$ by Step 2. Thus $u_{i^*} \in S'$ must be adjacent to $w_{p,q}$ which implies that G_{i^*} contains the edge $\{p,q\}$. Thus S is a clique in G_{i^*}.

Since we proved that the instance (G', Z', ℓ', k') can be constructed in polynomial-time and that it acts as the OR of the input instances, and because the parameter value k' is bounded by a polynomial in the size of the largest input instance, this concludes the cross-composition proof and establishes the claim. ◀

▶ **Corollary 12.** *If $\mathcal{F}$ is a class of graphs containing all cliques, then* VERTEX COVER *and* INDEPENDENT SET *parameterized by the minimum number of vertex deletions to obtain a graph in $\mathcal{F}$ do not admit polynomial kernels unless NP $\subseteq$ coNP/poly. In particular,* VERTEX COVER *and* INDEPENDENT SET *parameterized by co-cluster deletion distance or cluster deletion distance do not admit polynomial kernels unless NP $\subseteq$ coNP/poly.* ◀

4.2 Chromatic Number parameterized by Vertex Cover

In this section we give a kernelization lower bound for CHROMATIC NUMBER parameterized by vertex cover, through the use of a restricted version of 3-COLORING.

▶ **Definition 13.** A graph G is a *triangle split graph* if $V(G)$ can be partitioned into sets X, Y such that $G[X]$ is an edgeless graph and $G[Y]$ is a disjoint union of vertex-disjoint triangles.

An instance of the classical problem 3-COLORING WITH TRIANGLE SPLIT DECOMPOSITION is a tuple (G, X, Y) consisting of a graph G and a partition of its vertex set into $X \cup Y$ such that $G[X]$ is edgeless and $G[Y]$ is a union of vertex-disjoint triangles. The question is whether G has a proper 3-coloring. It is not hard to show show that this restricted form of the problem is NP-complete, by replacing all edges in a normal instance of 3-COLORING with a triangle.

▶ **Theorem 14.** CHROMATIC NUMBER *parameterized by the size of a vertex cover does not admit a polynomial kernel unless NP $\subseteq$ coNP/poly.*

Proof. To prove the theorem we will show that 3-COLORING WITH TRIANGLE SPLIT DECOMPOSITION cross-composes into CHROMATIC NUMBER parameterized by a vertex cover of the graph. By a suitable choice of polynomial equivalence relation in the same style as in Theorem 11 we may assume that we are given t input instances which encode structures $(G_1, X_1, Y_1), \ldots, (G_t, X_t, Y_t)$ of 3-COLORING WITH TRIANGLE SPLIT DECOMPOSITION with $|X_i| = n$ and $|Y_i| = 3m$ for all $i \in [t]$ (i.e., m is the number of triangles in each instance). We will compose these instances into one instance (G', Z', ℓ', k') of CHROMATIC NUMBER parameterized by vertex cover. By duplicating some instances we may assume that the number of inputs t is a power of 2; this only increases the input size by a factor of at most 2, and hence any bounds which are polynomial in the old input size will be polynomial in the new input size which is sufficient for our purposes.

For each set Y_i, label the triangles in $G_i[Y_i]$ as $T_1, \ldots, T_m$ in some arbitrary way, and label the vertices in each triangle T_j for a set Y_i as a_i^j, b_i^j, c_i^j. We build a graph G' with a vertex cover of size $k' := 3 \log t + 4 + 3m \in O(m + \log t)$ such that G' can be $\ell' := \log t + 4$-colored if and only if one of the input instances can be 3-colored.

1. Create a clique on vertices $\{p_i \mid i \in [\log t]\} \cup \{w, x, y, z\}$; it is called the *palette*.
2. Add the vertices $\bigcup_{i=1}^{t} X_i$ to the graph, and make them adjacent to the vertex w.
3. For $i \in [m]$ add a triangle T_i^* to the graph on vertices $\{a_i, b_i, c_i\}$. The union of these triangles will be the *triangle vertices* T^*. Make all vertices in T^* adjacent to all vertices from the set $\{p_i \mid i \in [\log t]\} \cup \{w\}$.
4. For $i \in [\log t]$ add a path on two new vertices $\{q_0^i, q_1^i\}$ to the graph, and make them adjacent to all vertices $(\{p_j \mid j \in [\log t]\} \cup \{x, y, z\}) \setminus \{p_i\}$. These vertices form the *instance selector* vertices.
5. For each instance number $i \in [t]$ consider the binary representation of the value i, which can be expressed in $\log t$ bits. Consider each position $j \in [\log t]$ of this binary representation, where position 1 is most significant and $\log t$ is least significant. If bit number j of the representation of i is a 0 (resp. a 1) then make vertex q_0^j (resp. q_1^j) adjacent to all vertices of X_i. (We identify t by the all-zero string $0 \ldots 0$.)
6. As the final step we re-encode the adjacencies between vertices in the independent sets X_i and the triangles into our graph G'. For each $i \in [t]$, for each vertex $v \in Y_i$, do the following. If v is adjacent in G_i to vertex a_i^j then make vertex v adjacent in G' to a_j. Do the same for adjacencies of v to b_i^j and c_i^j.

This concludes the construction. The following claims about G' are easy to verify:

(I) In every proper $\ell' = \log t + 4$-coloring of G', the following must hold:

   a. each of the $\log t + 4$ vertices of the palette clique receives a unique color,
   b. consider some $i \in [\log t]$: the vertices q_0^i and q_1^i receive different colors (since they are adjacent), one of them must take the color of w and the other of p_i (they are adjacent to all other vertices of the palette),
   c. the triangle vertices T^* are colored using the colors of x, y, z (they are adjacent to all other vertices of the palette),
   d. the only colors which can occur on a vertex in X_i (for all $i \in [t]$) are the colors given to x, y, z and $\{p_j \mid j \in [\log t]\}$ (since the vertices in X_i are adjacent to w).

(II) For every $i \in [t]$, the graph $G'[X_i \cup T^*]$ is isomorphic to G_i.

(III) The set $Z' := \{p_i \mid i \in [\log t]\} \cup \{w, x, y, z\} \cup T^* \cup \{q_0^i, q_1^i \mid i \in [\log t]\}$ forms a vertex cover of G' of size $k' = |Z'| = 3 \log t + 4 + 3m$. Hence we establish that G' has a vertex cover of size $O(m + \log t)$.

Using the given properties of G' one may verify that $\chi(G') \leq \log t + 4 \Leftrightarrow \exists i \in [t] : \chi(G_i) \leq 3$. The remainder of the proof is deferred to the full version due to space restrictions. ◀

For every fixed integer q, the q-Coloring problem parameterized by the vertex cover number *does* admit a polynomial kernel. This fact was independently observed by one of the referees. Kernelization algorithms for structural parameterizations of the q-Coloring problem will be the topic of a future publication.

4.3 Kernelization lower bounds for Feedback Vertex Set

In this section we give several kernelization lower bounds for Feedback Vertex Set. The proofs can be found in the full version [5].

▶ **Theorem 15.** *Unless NP ⊆ coNP/poly,* Feedback Vertex Set *does not admit a polynomial kernel when parameterized by 1) deletion distance to cluster graphs, and 2) deletion distance to co-cluster graphs.* ◀

▶ **Theorem 16.** Weighted Feedback Vertex Set*, where each vertex is given a positive integer as its weight, does not admit a polynomial kernel parameterized by the size of a vertex cover unless NP ⊆ coNP/poly.* ◀

5 Conclusions

We have introduced the technique of cross-composition and used it to derive kernelization lower bounds for structural parameterizations of several graph problems. Since we expect that cross-composition will be a fruitful tool in the further study of kernelization lower bounds, we give some pointers on how to devise cross-composition constructions. As the source problem of the composition one may choose a restricted yet NP-hard version of the target problem; this brings down the richness of the instances that need to be composed. If the goal is to give a lower bound for a structural parameterization (such as the size of a vertex cover) then starting from a problem on graphs which decompose into an independent set and some very structured remainder (e.g. triangle split graphs decompose into an independent set and vertex-disjoint triangles) it may be possible to compose the instances by taking the disjoint union of the inputs, and one-by-one identifying the vertices in the structured remainder. The fact that cross-compositions allow the output parameter to be polynomial in the size of the largest input can also be exploited, e.g., the proof of Theorem 11 uses this when composing input instances on n vertices into a graph G': we create $n^{O(1)}$ vertices inside a vertex cover Z' for G', and the adjacencies between Z' and a single vertex outside the cover represent the entire adjacency structure of an input graph.

Cross-composition is also appealing from a methodological point of view, since it gives a unified way of interpreting the two earlier techniques for proving kernelization lower bounds: or-compositions and polynomial-parameter transformations can both be seen to yield cross-compositions for a problem. For or-composition this is trivial to see since an or-composition for problem Q just shows that the unparameterized variant $\tilde{Q}$ cross-composes into Q. The combination of an or-composition for problem P and a polynomial-parameter transform from P to Q also gives a cross-composition: first applying the or-composition on instances of P and then transforming the resulting P-instance to a Q-instance effectively shows that we can cross-compose instances of the unparameterized variant $\tilde{P}$ into instances of Q. Hence the cross-composition technique puts the existing methods of showing super-polynomial kernelization lower bounds in a common framework, and also explains *why* these problems do not admit polynomial kernels: a parameterized problem P does not admit a polynomial kernel if it can encode the or of *some* NP-hard problem for a sufficiently small parameter value. This new perspective might lead to a deeper insight into the common structure of FPT problems without polynomial kernels.

Acknowledgements We would like to thank Holger Dell for insightful discussions which led to a more elegant proof of Theorem 9.

References

1 Peter Allen, Vadim Lozin, and Michaël Rao. Clique-width and the speed of hereditary properties. *The Electronic Journal of Combinatorics*, 16(1), 2009.

2 Hans L. Bodlaender. Kernelization: New upper and lower bound techniques. In *Proc. 4th IWPEC*, pages 17–37, 2009.

3 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

4 Hans L. Bodlaender, Fedor V. Fomin, Daniel Lokshtanov, Eelko Penninkx, Saket Saurabh, and Dimitrios M. Thilikos. (Meta) Kernelization. In *Proc. 50th FOCS*, pages 629–638, 2009.

5 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Cross-composition: A new technique for kernelization lower bounds. *CoRR*, abs/1011.4224, 2010.

6 Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *Comput. J.*, 51(3):255–269, 2008.

7 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. In *Proc. 17th ESA*, pages 635–646, 2009.

8 Jin-yi Cai, Venkatesan T. Chakaravarthy, Lane A. Hemaspaandra, and Mitsunori Ogihara. Competing provers yield improved Karp-Lipton collapse results. *Inf. Comput.*, 198(1):1–23, 2005.

9 Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003.

10 Yixin Cao, Jianer Chen, and Yang Liu. On feedback vertex set new measure and new structures. In *Proc. 12th SWAT*, pages 93–104, 2010.

11 Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

12 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *Proc. 42nd STOC*, pages 251–260, 2010.

13 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and IDs. In *Proc. 36th ICALP*, pages 378–389, 2009.

14 Rod Downey and Michael R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, New York, 1999.

15 Henning Fernau, Fedor V. Fomin, Daniel Lokshtanov, Daniel Raible, Saket Saurabh, and Yngve Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. In *Proc. 26th STACS*, Dagstuhl, Germany, 2009.

16 J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series).* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

17 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. Syst. Sci.*, 77(1):91–106, 2011.

18 Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

19 Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007.

20 Gregory Gutin, Eun Jung Kim, Stefan Szeider, and Anders Yeo. Solving max-2-sat above a tight lower bound. *CoRR*, abs/0907.4573, 2009.

21 Gregory Gutin, Leo van Iersel, Matthias Mnich, and Anders Yeo. All ternary permutation constraint satisfaction problems parameterized above average have kernels with quadratic numbers of variables. In *Proc. 18th ESA*, pages 326–337, 2010.

22 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

23 Stéphan Thomassé. A quadratic kernel for feedback vertex set. *ACM Transactions on Algorithms*, 6(2), 2010.

24 Chee-Keng Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26:287–300, 1983.

Vertex Cover Kernelization Revisited: Upper and Lower Bounds for a Refined Parameter*

Bart M. P. Jansen[1] and Hans L. Bodlaender[1]

1 **Department of Information and Computing Sciences, Utrecht University**
P.O. Box 80.089, 3508 TB, Utrecht, The Netherlands
`{bart,hansb}@cs.uu.nl`

Abstract

Kernelization is a concept that enables the formal mathematical analysis of data reduction through the framework of parameterized complexity. Intensive research into the Vertex Cover problem has shown that there is a preprocessing algorithm which given an instance (G, k) of Vertex Cover outputs an equivalent instance (G', k') in polynomial time with the guarantee that G' has at most $2k'$ vertices (and thus $O((k')^2)$ edges) with $k' \leq k$. Using the terminology of parameterized complexity we say that k-Vertex Cover has a kernel with $2k$ vertices. There is complexity-theoretic evidence that both $2k$ vertices and $\Theta(k^2)$ edges are optimal for the kernel size. In this paper we consider the Vertex Cover problem with a different parameter, the size $\textsc{fvs}(G)$ of a minimum feedback vertex set for G. This refined parameter is structurally smaller than the parameter k associated to the vertex covering number $\textsc{vc}(G)$ since $\textsc{fvs}(G) \leq \textsc{vc}(G)$ and the difference can be arbitrarily large. We give a kernel for Vertex Cover with a number of vertices that is cubic in $\textsc{fvs}(G)$: an instance (G, X, k) of Vertex Cover, where X is a feedback vertex set for G, can be transformed in polynomial time into an equivalent instance (G', X', k') such that $k' \leq k$, $|X'| \leq |X|$ and most importantly $|V(G')| \leq 2k$ and $|V(G')| \in O(|X'|^3)$. A similar result holds when the feedback vertex set X is not given along with the input. In sharp contrast we show that the Weighted Vertex Cover problem does not have a polynomial kernel when parameterized by $\textsc{fvs}(G)$ unless the polynomial hierarchy collapses to the third level ($\text{PH} = \Sigma_3^p$). Our work is one of the first examples of research in kernelization using a non-standard parameter, and shows that this approach can yield interesting computational insights. To obtain our results we make extensive use of the combinatorial structure of independent sets in forests.

1998 ACM Subject Classification F.2.2

Keywords and phrases kernelization, lower bounds, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.177

1 Introduction

The Vertex Cover problem is one of the six classic NP-complete problems discussed by Garey and Johnson in their famous work on intractability [22, GT1], and has played an important role in the development of parameterized algorithms [15, 28, 16]. A parameterized problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, and such a problem is (strongly uniform) *fixed parameter tractable* if membership of an instance (x, k) can be decided in $f(k)|x|^c$ time for some computable function f and constant c. Since the structure of Vertex Cover is so simple and elegant, it has proven to be an ideal testbed for new techniques in the context of

* This work was supported by the Netherlands Organisation for Scientific Research (NWO), project “KERNELS: Combinatorial Analysis of Data Reduction”.

28th Symposium on Theoretical Aspects of Computer Science (STACS’11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 177–188
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

parameterized complexity. The problem is also highly relevant from a practical point of view because of its role in bioinformatics [1] and other problem areas.

In this work we suggest a "refined parameterization" for the Vertex Cover problem using the feedback vertex number $\textsc{fvs}(G)$ as the parameter, i.e. the size of a smallest vertex set whose deletion turns G into a forest. We give upper bounds on the kernel size for the unweighted version of Vertex Cover under this parameterization, and also supply a conditional superpolynomial lower bound on the kernel size for the variant of Vertex Cover where each vertex has a non-negative integral weight. But before we state our results we shall first survey the current state of the art for the parameterized analysis of Vertex Cover.

There has been an impressive series of ever-faster parameterized algorithms to solve k-Vertex Cover, which led to the current-best algorithm by Chen et al. that can decide whether a graph G has a vertex cover of size k in $O(1.2738^k + kn)$ time and polynomial space [9, 30, 8, 17]. The Vertex Cover problem has also played an important role in the development of *problem kernelization* [23]. A kernelization algorithm (or *kernel*) is a polynomial-time procedure that reduces an instance (x, k) of a parameterized decision problem to an equivalent instance (x', k') such that $|x'|, k' \leq f(k)$ for some computable function f, which is the *size* of the kernel. We also use the term kernel to refer to the reduced instance (x', k').

The k-Vertex Cover problem admits a kernel with $2k$ vertices and $O(k^2)$ edges, which has been a subject of repeated study [6, 8, 10, 2, 11] and experimentation [1, 13]. There is some complexity-theoretic evidence that the size bounds for the kernel cannot be improved. Since practically all reduction-rules found to date are approximation-preserving [28], it appears that a kernel with less than $2k$ vertices would yield a polynomial-time approximation algorithm with a performance ratio smaller than 2 which would disprove the Unique Games Conjecture [25]. A recent breakthrough result by Dell and Van Melkebeek [12] shows that there is no polynomial kernel which can be encoded into $O(k^{2-\epsilon})$ bits for any $\epsilon > 0$ unless the polynomial hierarchy collapses to the third level ($\mathrm{PH} = \Sigma_3^p$), which suggests that the current bound of $O(k^2)$ edges is tight up to logarithmic factors.

This overview might suggest that there is little left to explore concerning kernelization for vertex cover, but this is far from true. All existing kernelization results for Vertex Cover use the requested size k of the vertex cover as the parameter. But there is no reason why we should not consider structurally smaller parameters, to see if we can preprocess instances of Vertex Cover such that their final size is bounded by a function of such a smaller parameter, rather than by a function of the requested set size k. We study kernelization for the Vertex Cover problem using the feedback vertex number $\textsc{fvs}(G)$ as the parameter. Since every vertex cover is also a feedback vertex set we find that $\textsc{fvs}(G) \leq \textsc{vc}(G)$ which shows that the feedback vertex number of a graph is a *structurally smaller* parameter than the vertex covering number: there are trees with arbitrarily large values of $\textsc{vc}(G)$ for which $\textsc{fvs}(G) = 0$. We call our parameter "refined" since it is structurally smaller than the standard parameter for the Vertex Cover problem.

Related Work. The idea of studying parameterized problems using alternative parameters is not new (see e.g. [28]), but was recently advocated by Fellows et al. [19, 20, 29] in the call to investigate the *complexity ecology* of parameters. The main idea behind this program is to determine how different parameters affect the parameterized complexity of a problem. Some recent results in this direction include FPT algorithms for graph layout problems parameterized by the vertex cover number of the graph [21] and an algorithm to decide isomorphism on graphs of bounded feedback vertex number [26]. We are aware of only two applications of this idea to give polynomial kernels using alternative parameters. Fellows et al. [20, 18] show that the problems Independent Set, Dominating Set and Hamiltonian Circuit admit linear-vertex kernels on graphs G when parameterized by the maximum number of leaves in any spanning tree of G. Very recently Uhlmann and Weller [31] gave a polynomial kernel for Two-Layer Planarization parameterized by the feedback edge set number, which is a refined structural parameter for that problem since it is smaller than the natural parameter.

Our Results. We believe that we are one of the first to present a polynomial problem kernel using a non-standard but practically relevant refined parameter. We study the following parameterized problem:

fvs-Weighted Vertex Cover
Instance: A simple undirected graph G, a weight function $w : V(G) \to \mathbb{N}^+$, a feedback vertex set $X \subseteq V(G)$ such that $G - X$ is a forest, an integer $k \geq 0$.
Parameter: The size $|X|$ of the feedback vertex set.
Question: Is there a vertex cover C of G such that $\sum_{v \in C} w(v) \leq k$?

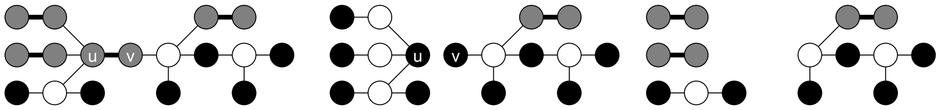
We also consider the unweighted variant fvs-Vertex Cover in which all vertices have a weight of 1. The problems fvs-Weighted Independent Set and fvs-Independent Set are defined similarly. Throughout this work k will always represent the total size or weight of the set we are looking for; depending on the context this is either a vertex cover or an independent set.

We prove that fvs-Vertex Cover has a kernel in which the number of vertices is bounded by $\min(O(|X|^3), 2k)$. This bound is at least as small as the current-best Vertex Cover kernel, but for graphs with small feedback vertex sets our bound is significantly smaller. We also study the weighted version of the problem, and obtain a contrasting result: we show that fvs-Weighted Vertex Cover does not admit a polynomial kernel unless $\mathrm{PH} = \Sigma_3^p$. This is very surprising since both the weighted and unweighted versions of k-Vertex Cover admit polynomial kernels and can be attacked using similar reduction rules [10]. To our knowledge we give the first example of a parameterized problem whose weighted and unweighted versions are both NP-complete and FPT, but for which the unweighted version allows a polynomial kernel but the weighted version does not.

When we present our results we will state them in terms of fvs-Independent Set and fvs-Weighted Independent Set since this simplifies the exposition. Because we are using the size of a feedback vertex set as the parameter, there are trivial parameterized reductions between these problems: an instance (G, X, k) of fvs-Vertex Cover is equivalent to an instance $(G, X, |V(G)| - k)$ of fvs-Independent Set with the same parameter value $|X|$. Hence our kernelization bounds for Independent Set carry over to Vertex Cover.

2 Preliminaries

In this work we only consider undirected, finite, simple graphs. Let G be a graph and denote its vertex set by $V(G)$ and the edge set by $E(G)$. We denote the independence number of G by $\alpha(G)$, the vertex covering number by $\textsc{vc}(G)$ and the feedback vertex number by $\textsc{fvs}(G)$. We will abbreviate maximum independent set as MIS, and feedback vertex set as FVS. For $v \in V(G)$ we denote the open and closed neighborhoods of v by $N_G(v)$ and $N_G[v]$, respectively. For a set $S \subseteq V(G)$ we have $N_G(S) := \bigcup_{v \in S} N_G(v) \setminus S$, and $N_G[S] := \bigcup_{v \in S} N_G[v]$. We write $G' \subseteq G$ if G' is a subgraph of G. The graph $G[V(G) \setminus X]$ obtained from G by deleting the vertices in X and their incident edges is denoted by $G - X$. The graph $G[E(G) \setminus Y]$ obtained



(a) Graph G and its independence decomposition.
(b) Decomposition for graph $G/\{\{u,v\}\}$.
(c) Decomposition for graph $G-\{u,v\}$.

Figure 1 Examples of the independence decomposition of a graph. Black vertices are in A, white vertices are in N, gray vertices are in S and the edges in M are drawn with thick lines.

from G by deleting the edges in Y but *not* their endpoints is denoted by G/Y. Carefully observe the difference between these two operators: if $\{u,v\}$ is an edge in G, then $G-\{u,v\}$ is the graph obtained from G by deleting the vertices u,v and their incident edges, whereas $G/\{\{u,v\}\}$ is the graph obtained from G by removing the edge $\{u,v\}$ while leaving the endpoints u and v intact. We note that many details had to be omitted from this extended abstract due to space restrictions; they can be found in the full version [24] of this work.

We need the following proposition on the structure of maximum independent sets in trees by Zito [32, Theorem 2], which we re-state here in terms of forests:

▶ Proposition 1. Let F be a forest. Then there is a unique partition of the vertex set $V(F)$ into subsets A,N,S such that:

1. Any MIS for F contains all vertices of A and no vertices of N.
2. For each vertex $v \in S$ there is a MIS for F containing v and a MIS for F avoiding v.
3. There is a perfect matching M in $F[S]$, and any MIS for F contains exactly one endpoint of each edge in M.
4. The matching M contains all the α-critical edges of F: for all $e \in E(F)$ it holds that $\alpha(F) < \alpha(F/\{e\}) \Leftrightarrow e \in M$.

This partition is uniquely characterized by adjacency relations. The sets A,N,S form the described partition if and only if:

I. There is a matching M on the vertices of S.
II. No vertex of A is adjacent to another vertex of A or to a vertex in S.
III. Each vertex of N is adjacent to at least two vertices of A.

We will refer to this decomposition of the vertex set of a forest F into the subsets A,N,S with the matching M as its *independence decomposition* (Figure 1).

▶ Observation 1. Let G be a graph. If G' is a vertex-induced subgraph of G then $\alpha(G) \geq \alpha(G')$, so for all $W \subseteq V(G)$ we have $\alpha(G) \geq \alpha(G-W)$. If G'' is an edge-induced subgraph of G then $\alpha(G'') \geq \alpha(G)$, so for all $Z \subseteq E(G)$ we have $\alpha(G) \leq \alpha(G/Z)$.

▶ Observation 2. If G is a graph and v is a vertex in G such that $\deg_G(v) \leq 1$ then there is a MIS for G that contains v.

3 Cubic Kernel for FVS-Independent Set

In this section we develop a cubic kernel for FVS-Independent Set. Consider an instance (G,X,k) of the problem, which asks whether graph G with the FVS X has an independent set of size k. Throughout this section $F := G-X$ denotes the forest obtained by deleting the vertices in X. Our starting point is the current-best Vertex Cover kernelization [8, Theorem 2.2] which exploits a theorem by Nemhauser and Trotter [27].

▶ **Theorem 1.** *There is a polynomial-time algorithm that takes an instance (G, k) of* Vertex Cover *as input, and computes in polynomial time an equivalent instance (G', k') such that G' is a vertex-induced subgraph of G with $k' \leq k$, $|V(G')| - k' \leq |V(G)| - k$ and $|V(G')| \leq 2k'$. We can ensure that G' does not contain any vertices of degree ≤ 1.*

Through the correspondence between Vertex Cover and Independent Set we can use Theorem 1 to preprocess an instance (G, X, k) of fvs-Independent Set.

▶ Reduction Rule 1. Let (G, X, k) be the current instance of fvs-Independent Set. Run the algorithm from Theorem 1 on the Vertex Cover instance $(G, |V(G)| - k)$ and let the result be $(G', |V(G')| - k')$. Obtain X' from X by deleting the vertices that were removed from G by the algorithm, and use (G', X', k') as the new instance of fvs-Independent Set.

When given an independent subset $X' \subseteq X$ of the feedback vertices we can efficiently compute the largest independent set I in G which satisfies $I \cap X = X'$: since such a set intersects X exactly in X', and since it cannot use any neighbors of X' the maximum size is $|X'| + \alpha(F - N_G(X'))$ and this is polynomial-time computable since $F - N_G(X')$ is a forest. We exploit this to assess which vertices from the FVS X might occur in a MIS of G.

▶ **Definition 2.** The number of *conflicts* $\textsc{Conf}_{F'}(X')$ induced by a subset $X' \subseteq X$ on a subforest $F' \subseteq F \subseteq G$ is defined as $\textsc{Conf}_{F'}(X') := \alpha(F') - \alpha(F' - N_G(X'))$.

This term $\textsc{Conf}_{F'}(X')$ can be interpreted as follows. Choosing vertices from X' in an independent set will prevent all their neighbors in the subforest F' from being part of the same independent set; hence if we fix some choice of vertices in X', then the number of vertices from F' we can add to this set (while maintaining independence) might be smaller than the independence number of F'. The term $\textsc{Conf}_{F'}(X')$ measures the difference between the two: informally it is the price we pay in the forest F' for choosing the vertices X' in the independent set. We can now formulate our first new reduction rules.

▶ Reduction Rule 2. If there is a vertex $v \in X$ such that $\textsc{Conf}_F(\{v\}) \geq |X|$, then delete v from the graph G and from the set X.

▶ Reduction Rule 3. If there are distinct vertices $u, v \in X$ with $\{u, v\} \notin E(G)$ for which $\textsc{Conf}_F(\{u, v\}) \geq |X|$, then add the edge $\{u, v\}$ to G.

Correctness of these two rules can be established from the following lemma.

▶ **Lemma 3.** *If $X' \subseteq X$ is a subset of feedback vertices such that $\textsc{Conf}_F(X') \geq |X|$ then there is a MIS for G that does not contain all vertices of X'.*

Proof. Assume that $I \subseteq V(G)$ is an independent set containing all vertices of X'. We will prove that there is an independent set I' which is disjoint from X' with $|I'| \geq |I|$. Since $\textsc{Conf}_F(X') \geq |X|$ it follows by definition that $\alpha(F) - \alpha(F - N_G(X')) \geq |X|$; since I cannot contain any neighbors of vertices in X' we know that $|I \cap V(F)| \leq \alpha(F - N_G(X'))$, and since $|V(G)| = |X| + |V(F)|$ we have $|I| \leq |X| + \alpha(F - N_G(X')) \leq \alpha(F)$. Hence the maximum independent set for F, which does not contain any vertices of X', is at least as large as I; this proves that for every independent set containing X' there is another independent set which is at least as large and avoids the vertices of X'. Therefore there is a MIS for G avoiding at least one vertex of X'. ◀

▶ Reduction Rule 4. If F contains a connected component T (which must be a tree) such that for all $X' \subseteq X$ with $|X'| \leq 2$ for which X' is independent in G it holds that $\textsc{Conf}_T(X') = 0$, then delete T from graph G and decrease k by $\alpha(T)$.

To prove the correctness of Rule 4 we need the following lemma.

▶ **Lemma 4.** *Let T be a connected component of F and let $X_I \subseteq X$ be an independent set in G. If $\textsc{Conf}_T(X_I) > 0$ then there is a set $X' \subseteq X_I$ with $|X'| \leq 2$ such that $\textsc{Conf}_T(X') > 0$.*

Proof. Assume the conditions stated in the lemma hold. Consider the independence decomposition of T into the sets A, N, S, and let M be a perfect matching on $T[S]$. We will try to construct a MIS I for T that does not use any vertices in $N_G(X_I)$; this must then also be a MIS for $T - N_G(X_I)$ of the same size. By the assumption that $\textsc{Conf}_T(X_I) > 0$ any independent set in T must use at least one vertex in $N_G(X_I)$ in order to be maximum, hence our construction procedure must fail somewhere; the place where it fails will provide us with a set X' as required by the statement of the lemma.

Construction of a MIS. By Proposition 1 any MIS for T must use all vertices in A, no vertices from N and exactly one endpoint of each edge in the matching M. It follows that if some vertex $v \in A$ is adjacent in G to a vertex $x \in X_I$, then $\alpha(T - \{v\}) < \alpha(T)$ and therefore $\alpha(T - N_G(x)) < \alpha(T)$ which proves that $\textsc{Conf}_T(\{x\}) > 0$; hence we can then use $X' := \{x\}$ as our desired subset to prove the claim. In the remainder of the proof we may therefore assume that no vertex of A is adjacent in G to a vertex in X_I.

We now start building our independent set I for T that avoids vertices in $N_G(X_I)$. We start by taking all vertices of A in the independent set; we do not use any vertices in $N_G(X_I)$ here since $A \cap N_G(X_I) = \emptyset$ by assumption. To augment I into a MIS for T it remains to add one endpoint of each edge in the matching M. Since the endpoints of the matching are not adjacent to vertices in A by the adjacency rules of Proposition 1, we can now restrict ourselves to the subgraph $T' := T[S]$ induced by the matched vertices since no choice of independent vertices in $T[S]$ will conflict with the choice of the vertices A. If there is a vertex v in T' such that $N_{T'}(v) = \{u\}$ and $N_G(v) \cap X_I = \emptyset$, then the edge $\{v, u\}$ must be in the matching M (since M is a perfect matching in $T[S]$). Because we must choose one of $\{u, v\}$ in a MIS for T, and by Observation 2 choosing a degree-1 vertex will never conflict with choices that are made later on, we can add v to our independent set I while respecting the invariant that no vertex in I is adjacent in G to a vertex in X_I. Since we have then chosen one endpoint of the matching edge $\{u, v\}$ in I, we can delete u, v and their incident edges to obtain a smaller graph T' (which again contains a perfect submatching of M) in which we continue the process. As long as there is a vertex with degree 1 in T' that has no neighbors in X_I then we take it into I, delete it and its neighbor, and continue. If this process ends with an empty graph, then by Proposition 1 the set I must be a MIS for T, and since it does not use any vertices adjacent to X_I it must also be a MIS for $T - N_G(X_I)$; but this proves that $\alpha(T) = \alpha(T - N_G(X_I))$ which means $\textsc{Conf}_T(X_I) = 0$, which is a contradiction to the assumption at the start of the proof. So the process must end with a non-empty graph $T' \subseteq T$ such that vertices with degree 1 in T' are adjacent in G to a vertex in X_I and for which the matching M restricted to T' is a perfect matching on T'. We use this subgraph T' to obtain a set X' as desired.

Using the subgraph to prove the claim. Consider a vertex v_0 in T' with $\deg_{T'}(v_0) = 1$, and construct a path $P = \{v_0, v_1, \ldots, v_{2p+1}\}$ by following edges of T' that are alternatingly in and out of the matching M, until arriving at a degree-1 vertex whose only neighbor was already visited. Since T' is acyclic, M restricted to T' is a perfect matching on T' and we start the process at a vertex of degree 1, it is easy to verify that there must be such a path P (there can be many; any arbitrary such path will suffice), that P must contain an even number of vertices, that the first and last vertex on P have degree-1 in T' and that the edges $\{v_{2i}, v_{2i+1}\}$ must be in M for all $0 \leq i \leq p$. Since we assumed that all degree-1 vertices in T' are adjacent in G to X_I, there exist vertices $x_1, x_2 \in X$ such that $v_0 \in N_G(x_1)$

and $v_{2p+1} \in N_G(x_2)$. We now claim that $X' := \{x_1, x_2\}$ satisfies the requirements of the statement of the lemma, i.e. that $\textsc{Conf}_T(\{x_1, x_2\}) > 0$. This fact is witnessed by considering the path P in the original tree T. Any MIS for T which avoids $N_G(\{x_1, x_2\})$ must use one endpoint of the matched edge $\{v_0, v_1\}$, and since the choice of v_0 is blocked because v_0 is a neighbor to x_1, it must use v_1. But path P shows that v_1 is adjacent in T to v_2, and hence we cannot choose v_2 in the independent set. But since $\{v_2, v_3\}$ is again a matched edge, we must use one of its endpoints; hence we must use v_3. Repeating this argument shows that we must use vertex v_{2p+1} in a MIS for T if we cannot use v_0; but the use of v_{2p+1} is also not possible if we exclude $N_G(\{x_1, x_2\})$. Hence we cannot make a MIS for T without using vertices in $N_G(\{x_1, x_2\})$ which proves that $\alpha(T) > \alpha(T - N_G(\{x_1, x_2\})$. By the definition of conflicts this proves that $\textsc{Conf}_T(X') > 0$ for $X' = \{x_1, x_2\}$, which concludes the proof. ◀

Using this lemma we can prove the correctness of Rule 4.

▶ **Lemma 5.** *Rule 4 is correct: if T is a connected component in F such that for all $X' \subseteq X$ which are independent in G and satisfy $|X'| \leq 2$ it holds that $\textsc{Conf}_T(X') = 0$, then $\alpha(G) = \alpha(G - T) + \alpha(T)$.*

Proof. Assume the conditions in the statement of the lemma hold. It is trivial to see that $\alpha(G) \leq \alpha(G - T) + \alpha(T)$. To establish the lemma we only need to prove that $\alpha(G) \geq \alpha(G - T) + \alpha(T)$, which we will do by showing that any independent set I_{G-T} in $G - T$ can be transformed to an independent set of size at least $|I_{G-T}| + \alpha(T)$ in G. So consider such an independent set I_{G-T}, and let $X_I := I_{G-T} \cap X$ be the set of vertices which belong to both I_{G-T} and the feedback vertex set X. Suppose that $\alpha(T) > \alpha(T - N_G(X_I))$. Then by Lemma 4 there is a subset $X' \subseteq X_I$ with $|X'| \leq 2$ such that $\textsc{Conf}_T(X') > 0$. Since X_I is an independent set, such a subset X' would also be independent; but by the preconditions to this lemma such a set X' does not exist and therefore we must have $\alpha(T) = \alpha(T - N_G(X_I))$.

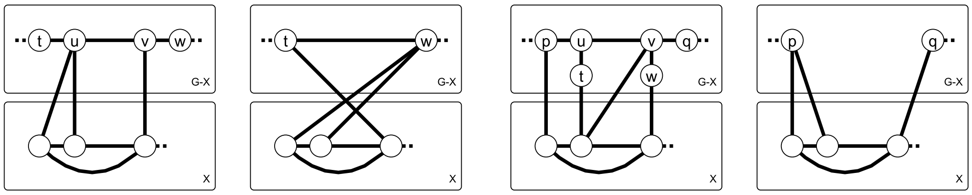
Now we show how to transform I_{G-T} into an independent set for G of the requested size. Let I_T be a MIS in $T - N_G(X_I)$, which has size $\alpha(T - N_G(X_I)) = \alpha(T)$. It is easy to verify that $I_{G-T} \cup I_T$ must be an independent set in G because vertices of T are only adjacent to vertices of $G - T$ which are contained in X. Hence the set $I_{G-T} \cup I_T$ is independent in G and it has size $|I_{G-T}| + \alpha(T)$. Since this argument applies to any independent set I_{G-T} in graph $G - T$ it holds in particular for a MIS in $G - T$, which proves that $\alpha(G) \geq \alpha(G - T) + \alpha(T)$ which proves the claim. ◀

We introduce the concept of blockability for the statement of the last two reduction rules.

▶ **Definition 6.** We say that the pair $x, y \in V(G) \setminus X$ is X-*blockable* if G contains an independent set $X' \subseteq X$ of size $|X'| \leq 2$ such that $\{x, y\} \subseteq N_G(X')$.

This can be interpreted as follows: any independent set in G that contains X' cannot contain x or y, so the pair x, y is blocked from being in an independent set by choosing X'. It follows directly from the definition that if x, y is not X-blockable, then for any combination of $u \in N_G(x) \cap X$ and $v \in N_G(y) \cap X$ we must have $\{u, v\} \in E(G)$.

▶ Reduction Rule 5. If there are distinct vertices $u, v \in V(G) \setminus X$ which are adjacent in G and are not X-blockable such that $\deg_F(u), \deg_F(v) \leq 2$ then reduce the graph as follows. Delete vertices u, v and decrease k by 1. If u has a neighbor t in F which is not v, then make all vertices of $N_G(v) \cap X$ adjacent to t. If v has a neighbor w in F which is not u, then make all vertices of $N_G(u) \cap X$ adjacent to w. If the vertices t, w exist then they must be unique; add the edge $\{t, w\}$ to the graph.



(a) Rule 5: Shrinking unblockable degree-2 paths in trees. ($k' := k - 1$)

(b) Rule 6: Removing unblockable pendants in trees. ($k' := k - 2$)

Figure 2 Illustrations of two reduction rules. The original structure is shown on the left, and the image on the right shows the structure after the reduction. Feedback vertices X are drawn in the bottom container, whereas the forest $G - X$ is visualized in the top container.

▶ Reduction Rule 6. If there are distinct vertices t, u, v, w in $V(G) \setminus X$ such that $\deg_F(u) = \deg_F(v) = 3$, $N_F(t) = \{u\}$, $N_F(w) = \{v\}$ and $\{u, v\} \in E(G)$ such that none of the pairs $\{u, t\}$, $\{v, w\}$, $\{t, w\}$ are X-blockable, then reduce as follows. Let $\{p\} = N_F(u) \setminus \{t, v\}$ and let $\{q\} = N_F(v) \setminus \{w, u\}$. Delete $\{t, u, v, w\}$ and their incident edges from G, decrease k by 2, make p adjacent to all vertices of $N_G(t) \cap X$ and make q adjacent to all vertices of $N_G(w) \cap X$.

See Figure 2 for an illustration of the final two reduction rules, which are meant to reduce the sizes of the trees in the forest F. The correctness of these rules can be proven by an exchange argument. Whereas Rule 4 deletes a tree T from the forest F when we can derive that for every independent set in $G - T$ we can obtain an independent set in G which is $\alpha(T)$ vertices larger, these last reduction rules act *locally* within one tree, but according to the same principle. Instead of working on an entire connected component of F, they reduce subtrees $T' \subseteq F$ in situations where we can derive that every independent set in X can be augmented with $\alpha(T')$ vertices from T'. In Rule 5 we reduce the subtree on vertices $\{u, v\}$ which has independence number 1, and in Rule 6 we reduce the subtree on vertices $\{u, v, t, w\}$ with independence number 2. Connections between the vertices adjacent to the reduced subtree are made to enforce that removal of the subtree does not affect the types of interactions between the neighboring vertices.

When no reduction rules can be applied to an instance, we call it *reduced*. In reduced instances the number of vertices in F must be bounded by a function of $|X|$, which can be proven using the following notion.

▶ **Definition 7.** Let F' be a subforest of F, and define the number of *active conflicts* induced on F' by the feedback set X as follows: $\textsc{Active}_{F'}(X) := \sum_{X' \in \mathcal{X}} \textsc{Conf}_{F'}(X')$ using the abbreviation $\mathcal{X} := \{X' \mid X' \subseteq X \wedge |X'| \leq 2 \wedge X' \text{ is independent in } G\}$.

The number of active conflicts induced on F in a reduced instance is polynomially bounded in $|X|$. For every $v \in X$ we have $\textsc{Conf}_F(\{v\}) < |X|$ by Rule 2, and every pair of distinct non-adjacent vertices $\{u, v\} \subseteq X$ satisfies $\textsc{Conf}_F(\{u, v\}) < |X|$ by Rule 3. Hence for every reduced instance we have $\textsc{Active}_F(X) \leq |X|^2 + \binom{|X|}{2}|X|$. A technical proof shows that in a reduced instance the number of active conflicts induced on the forest F is at least $\frac{1}{83}|V(F)|$. By combining this with the bound on the number of active conflicts, we can bound the size of reduced instances and obtain a kernelization algorithm. The algorithm exhaustively applies the six reduction rules, and the analysis then shows that the instance must be small when no more reduction rules can be applied. Using the duality of Vertex Cover and Independent Set we also obtain a kernel for fvs-Vertex Cover as a corollary.

▶ **Theorem 8.** FVS-INDEPENDENT SET *has a kernel with a cubic number of vertices: there is a polynomial-time algorithm that transforms an instance* (G, X, k) *into an equivalent instance* (G', X', k') *such that* $|X'| \leq |X|$, $k' \leq k$, $|V(G')| - k' \leq |V(G)| - k$, $|V(G')| \leq 2(|V(G')| - k')$ *and* $|V(G')| \leq |X| + 83|X|^3$.

▶ **Corollary 9.** FVS-VERTEX COVER *has a kernel with* $\min(2k, |X| + 83|X|^3)$ *vertices.*

4 No Polynomial Kernel for FVS-Weighted Independent Set

In this section we show that the introduction of vertex weights makes the parameterized INDEPENDENT SET problem harder to kernelize, by proving that FVS-WEIGHTED INDEPENDENT SET does not have a polynomial kernel unless PH $= \Sigma_3^p$. To establish this result, we introduce a new parameterized problem called t-PAIRED VECTOR AGREEMENT and show that it does not have a polynomial kernel unless PH $= \Sigma_3^p$. We then finish the proof by giving a polynomial-parameter transformation [5, 14] to FVS-WEIGHTED INDEPENDENT SET.

t-PAIRED VECTOR AGREEMENT
Instance: A list L consisting of t pairs of vectors (a^i, b^i) for $1 \leq i \leq t$ such that each vector is an element of $\{0, 1, \#, ?\}^m$, and an integer $k \geq 0$.
Parameter: The number of pairs t.
Question: Is it possible to choose one vector from each pair, such that the chosen vectors S induce at most k conflict positions? A position $1 \leq j \leq m$ in a vector is a *conflict* position if some chosen vector $v \in S$ has $v_j = \#$, or if we have chosen vectors $u, v \in S$ such that $u_j = 0$ and $v_j = 1$.

The framework for proving that a parameterized problem does not have a polynomial kernel unless PH $= \Sigma_3^p$ requires us to establish that the corresponding classical problem is NP-complete. A reduction from VERTEX COVER shows that the classic problem PAIRED VECTOR AGREEMENT is NP-complete. By exploiting the fact that t-PAIRED VECTOR AGREEMENT can be solved in $O(2^t p(m))$ time for some polynomial p (by trying all possible combinations of vectors), we can build an OR-composition algorithm for the paired agreement problem using a bitmask selection strategy; the techniques we use here are similar to those employed by Dom et al. [14]. These two facts prove that t-PAIRED VECTOR AGREEMENT does not have a polynomial kernel unless PH $= \Sigma_3^p$. To relate these results to FVS-WEIGHTED INDEPENDENT SET we use the following transformation.

▶ **Lemma 10.** *There is a polynomial-parameter reduction from* t-PAIRED VECTOR AGREEMENT *to* FVS-WEIGHTED INDEPENDENT SET.

Proof. Let (L, t, m, k) be an instance of t-PAIRED VECTOR AGREEMENT. We may assume that $k < m$ otherwise the answer to the instance is trivially YES. We show how to build an equivalent instance (G', w', X', k') of FVS-INDEPENDENT SET in polynomial time such that $|X'| = 2t$, which implies the existence of a polynomial-parameter reduction.

The graph G' has $2(t + m)$ vertices, and is defined as follows. For each index $1 \leq i \leq t$ there is a pair of vertices v_i^a, v_i^b which are connected by an edge, and have weight $2(t + m)$. For each vector position $1 \leq j \leq m$ there are vertices p_j^0, p_j^1 which are connected by an edge, and have weight 1. The vertices v_i^a and v_i^b correspond to the vectors a^i, b^i of the t-PAIRED VECTOR AGREEMENT instance, and are connected to the position-vertices as follows. Let v be a vertex v_i^a or v_i^b corresponding to the vector VEC(v) which is a_i or b_i, respectively. For $1 \leq i \leq t$ vertex v is adjacent in G' to all p_j^0 for which vector VEC(v) has a 0 at position j; it is also adjacent to all p_j^1 for which vector $\textsc{vec}(v)$ has a 1 at position j, and finally vertex v is adjacent to all $\{p_j^0, p_j^1\}$ for which vector $\textsc{vec}(v)$ has a $\#$ at position j. This concludes the definition of the structure of graph G'.

One may verify that a position vertex p_j^x is adjacent to exactly 1 other position vertex p_j^{1-x}, which implies that the graph induced by the position vertices $p_j^{0,1}$ has maximum degree 1 and is therefore a forest; this shows that the vector-vertices $v_i^{a/b}$ form a feedback vertex set for G' and thus we define the feedback vertex set for our instance as $X' := \{v_i^a, v_i^b \mid 1 \leq i \leq t\}$ which has size exactly $2t$. We now ask for an independent set of total weight at least $k' := 2t(t+m) + (m-k)$, which completes the description of instance (G', w', X', k'). It is easy to see that this instance can be computed in polynomial time from the instance (L, t, m, k). The proof that these two instances are equivalent is not difficult, and has been deferred to the full version of this paper. ◀

By standard kernelization lower-bound techniques (see [5, 14]) Lemma 10 implies:

▶ **Theorem 11.** *The problems* FVS-WEIGHTED INDEPENDENT SET *and* FVS-WEIGHTED VERTEX COVER *do not admit polynomial kernels unless PH* $= \Sigma_3^p$.

It is interesting to note that an instance (G', w', X', k') of FVS-INDEPENDENT SET resulting from the polynomial-parameter transformation of Lemma 10 has a very restricted graph structure: every connected component of the forest $G' - X'$ is a path on two vertices. Hence our proof shows that even using the parameter "number of vertex deletions needed to turn the graph into a disjoint union of P_2's" (a structurally *larger* parameter than the FVS size) there is no polynomial kernel unless PH $= \Sigma_3^p$.

5 Conclusion

We have given upper and lower bounds on the size of kernels for the VERTEX COVER and INDEPENDENT SET problems using the parameter $\textsc{fvs}(G)$. It would be very interesting to perform experiments with our new reduction rules to see whether they offer significant benefits over the existing VERTEX COVER kernel on real-world instances. This result is one of the first examples of a polynomial kernel using a "refined" parameter which is structurally smaller than the standard parameterization. The contrasting result on the weighted problem shows that there is a rich structure waiting to be uncovered when studying kernelization using non-standard parameters. The kernel we have presented for FVS-VERTEX COVER contains $O(|X|^3)$ vertices and can therefore be encoded in $O(|X|^6)$ bits using an adjacency matrix. The results of Dell and Van Melkebeek [12] imply that it is unlikely that there exists a kernel which can be encoded in $O(|X|^{2-\epsilon})$ bits for any $\epsilon > 0$. It might be possible to improve the size of the kernel to a quadratic or even a linear number of vertices, by employing new reduction rules. The current reduction rules can be seen as analogs of the traditional "high degree" rule for the VERTEX COVER problem, and it would be interesting to see whether it is possible to find analogs of crown reduction rules when using $\textsc{fvs}(G)$ as the parameter.

Although we have assumed throughout the paper that a feedback vertex set is supplied with the input, we can drop this restriction by applying the known polynomial-time 2-approximation algorithm for FVS [3]. Observe that the reduction algorithm does not require that the supplied set X is a *minimum* feedback vertex set; the kernelization algorithm works if X is *any* feedback vertex set, and the size of the output instance depends on the size of the FVS that is supplied. Hence if we compute a 2-approximate FVS and supply it to the kernelization algorithm, the bound on the number of vertices in the output instance is only a factor 2 worse than when running the kernelization using a *minimum* FVS.

This paper has focused on the decision version of the Vertex Cover problem, but the data reduction rules given here can also be translated to the optimization version to obtain the following result: given a graph G there is a polynomial-time algorithm that computes a graph G' and a non-negative integer c such that $\textsc{vc}(G) = \textsc{vc}(G') + c$ with $|V(G')| \leq 2\,\textsc{vc}(G)$ and $|V(G')| \in O(\textsc{fvs}(G)^3)$; and a vertex cover S' for G' can be transformed back into a vertex cover of G of size $|S'| + c$ in polynomial time.

The approach of studying Vertex Cover parameterized by $\textsc{fvs}(G)$ fits into the broad context of "parameterizing away from triviality" [28, 7], since the parameter $\textsc{fvs}(G)$ measures how many vertex-deletions are needed to reduce G to a forest in which Vertex Cover can be solved trivially in polynomial time. As there is a wide variety of restricted graph classes for which Vertex Cover is in P, this opens up a multitude of possibilities for non-standard parameterizations. For every graph class $\mathcal{G}$ which is closed under vertex deletion and for which the Vertex Cover problem is in P, the Vertex Cover problem is in FPT when parameterized by the size of a set X such that $G - X \in \mathcal{G}$, assuming that X is given as part of the input. Recent work [4] into this direction shows that whenever $\mathcal{G}$ contains all cliques the resulting parameterized problem does not have a polynomial kernel unless $\text{PH} = \Sigma_3^p$. Examples of such classes $\mathcal{G}$ are chordal graphs, interval graphs and other types of perfect graphs. We conclude with two specific open problems. Is there a polynomial kernel using the deletion distance from a bipartite graph as the parameter? Does the Vertex Cover problem parameterized by the size of a minimum set X such that $\textsc{treewidth}(G - X) \leq i$ have a polynomial kernel for every fixed i, or is there some value of i for which this problem does not have a polynomial kernel? The classic Vertex Cover kernelizations can be interpreted as the case $i = 0$ whereas this paper supplies the result for $i = 1$. It appears that many interesting insights are waiting to be discovered in this direction.

References

1 Faisal N. Abu-Khzam, Rebecca L. Collins, Michael R. Fellows, Michael A. Langston, W. Henry Suters, and Christopher T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *Proc. 6th ALENEX/ANALC*, pages 62–69, 2004.

2 Faisal N. Abu-Khzam, Michael R. Fellows, Michael A. Langston, and W. Henry Suters. Crown structures for vertex cover kernelization. *Theory Comput. Syst.*, 41(3):411–430, 2007.

3 Vineet Bafna, Piotr Berman, and Toshihiro Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999.

4 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Cross-composition: A new technique for kernelization lower bounds. *CoRR*, abs/1011.4224, 2010.

5 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. In *Proc. 17th ESA*, pages 635–646, 2009.

6 Jonathan F. Buss and Judy Goldsmith. Nondeterminism within P. *SIAM J. Comput.*, 22(3):560–572, 1993.

7 Leizhen Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003.

8 Jianer Chen, Iyad A. Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

9 Jianer Chen, Iyad A. Kanj, and Ge Xia. Improved parameterized upper bounds for vertex cover. In *Proc. 31st MFCS*, pages 238–249, 2006.

10 Miroslav Chlebík and Janka Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discrete Applied Mathematics*, 156(3):292–312, 2008.

11 Benny Chor, Mike Fellows, and David W. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *Proc. 30th WG*, pages 257–269, 2004.

12 Holger Dell and Dieter van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *Proc. 42nd STOC*, pages 251–260, 2010.

13 Josep Díaz, Jordi Petit, and Dimitrios M. Thilikos. Kernels for the vertex cover problem on the preferred attachment model. In *Proc. 5th WEA*, pages 231–240, 2006.

14 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and IDs. In *Proc. 36th ICALP*, pages 378–389, 2009.

15 Rod Downey and Michael R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, New York, 1999.

16 Rodney G. Downey, Michael R. Fellows, and Michael A. Langston, editors. *The Computer Journal: Special Issue on Parameterized Complexity*, volume 51, 2008.

17 Rodney G. Downey, Michael R. Fellows, and Ulrike Stege. Parameterized complexity: A framework for systematically confronting computational intractability. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 49–99, 1997.

18 Vladimir Estivill-Castro, Michael Fellows, Michael Langston, and Frances Rosamond. FPT is P-time extremal structure I. In *Proc. 1st ACiD*, pages 1–41, 2005.

19 Michael R. Fellows. Towards fully multivariate algorithmics: Some new results and directions in parameter ecology. In *Proc. 20th IWOCA*, pages 2–10, 2009.

20 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Matthias Mnich, Frances A. Rosamond, and Saket Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory Comput. Syst.*, 45(4):822–848, 2009.

21 Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *Proc. 19th ISAAC*, pages 294–305, 2008.

22 Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

23 Jiong Guo and Rolf Niedermeier. Invitation to data reduction and problem kernelization. *SIGACT News*, 38(1):31–45, 2007.

24 Bart M. P. Jansen and Hans L. Bodlaender. Vertex cover kernelization revisited: Upper and lower bounds for a refined parameter. *CoRR*, abs/1012.4701, 2010.

25 Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within $2-\epsilon$. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

26 Stefan Kratsch and Pascal Schweitzer. Isomorphism for graphs of bounded feedback vertex set number. In *Proc. 12th SWAT*, pages 81–92, 2010.

27 G.L. Nemhauser and L.E.jun. Trotter. Vertex packings: structural properties and algorithms. *Math. Program.*, 8:232–248, 1975.

28 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

29 Rolf Niedermeier. Reflections on multivariate algorithmics and problem parameterization. In *Proc. 27th STACS*, pages 17–32, 2010.

30 Rolf Niedermeier and Peter Rossmanith. On efficient fixed-parameter algorithms for weighted vertex cover. *J. Algorithms*, 47(2):63–77, 2003.

31 Johannes Uhlmann and Mathias Weller. Two-layer planarization parameterized by feedback edge set. In *Proc. 7th TAMC*, pages 431–442, 2010.

32 Jennifer Zito. The structure and maximum number of maximum independent sets in trees. *J. Graph Theory*, 15(2):207–221, 1991.

Hitting forbidden minors: Approximation and Kernelization

Fedor V. Fomin[1], Daniel Lokshtanov[2], Neeldhara Misra[3], Geevarghese Philip[3], and Saket Saurabh[3]

1 **Department of Informatics, University of Bergen, N-5020 Bergen, Norway.**
`fomin@ii.uib.no`

2 **University of California, San Diego, Department of Computer Science and Engineering, 9500 Gilman Drive, La Jolla, CA 92093-0404, USA,**
`dlokshtanov@cs.ucsd.edu`

3 **The Institute of Mathematical Sciences, Chennai 600113, India.**
`{neeldhara|gphilip|saket}@imsc.res.in`

Abstract

We study a general class of problems called p-$\mathcal{F}$-Deletion problems. In an p-$\mathcal{F}$-Deletion problem, we are asked whether a subset of at most k vertices can be deleted from a graph G such that the resulting graph does not contain as a minor any graph from the family $\mathcal{F}$ of forbidden minors. We obtain a number of algorithmic results on the p-$\mathcal{F}$-Deletion problem when $\mathcal{F}$ contains a planar graph. We give

- a linear vertex kernel on graphs excluding t-claw $K_{1,t}$, the star with t leves, as an induced subgraph, where t is a fixed integer.
- an approximation algorithm achieving an approximation ratio of $O(\log^{3/2} OPT)$, where OPT is the size of an optimal solution on general undirected graphs.

Finally, we obtain polynomial kernels for the case when $\mathcal{F}$ only contains graph θ_c as a minor for a fixed integer c. The graph θ_c consists of two vertices connected by c parallel edges. Even though this may appear to be a very restricted class of problems it already encompasses well-studied problems such as Vertex Cover, Feedback Vertex Set and Diamond Hitting Set. The generic kernelization algorithm is based on a non-trivial application of protrusion techniques, previously used only for problems on topological graph classes.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.189

1 Introduction

Let $\mathcal{F}$ be a finite set of graphs. In an p-$\mathcal{F}$-Deletion problem, we are given an n-vertex graph G and an integer k as input, and asked whether at most k vertices can be deleted from G such that the resulting graph does not contain a graph from $\mathcal{F}$ as a minor. We refer to such a subset as an $\mathcal{F}$-hitting set. The p-$\mathcal{F}$-Deletion problem is a generalization of several fundamental problems. For example, when $\mathcal{F} = \{K_2\}$, a complete graph on two vertices, this is the Vertex Cover problem. When $\mathcal{F} = \{C_3\}$, a cycle on three vertices, this is the Feedback Vertex Set problem. Other famous cases are $\mathcal{F} = \{K_{2,3}, K_4\}$, $\mathcal{F} = \{K_{3,3}, K_5\}$ and $\mathcal{F} = \{K_3, T_2\}$, which correspond to removing vertices to obtain outerplanar graphs, planar graphs, and graphs of pathwidth one respectively. Here, $K_{i,j}$ denotes the complete bipartite graph with bipartitions of sizes i and j, and K_i denotes the complete graph on i vertices. Further, a T_2 is a star on three leaves, each of whose edges has been subdivided exactly once. A T_2 structure is depicted in the leftmost graph of Figure 1.

Our interest in the p-$\mathcal{F}$-Deletion problem is motivated by its generality and the recent developments in kernelization or polynomial time preprocessing. The parameterized

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 189–200
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

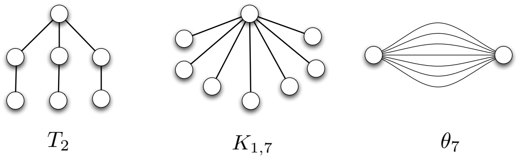


Figure 1 Graphs T_2, t-claw $K_{1,t}$ with $t = 7$, and θ_c with $c = 7$

complexity of this general problem is well understood. By a celebrated result of Robertson and Seymour, every p-$\mathcal{F}$-Deletion problem is non-uniformly fixed-parameter tractable (FPT). That is, for every k there is an algorithm solving the problem in time $O(f(k) \cdot n^3)$ [32]. In this paper we study this problem from the view point of polynomial time preprocessing and approximation, when the obstruction set $\mathcal{F}$ satisfies certain properties.

Preprocessing as a strategy for coping with hard problems is universally applied in practice and the notion of *kernelization* provides a mathematical framework for analyzing the quality of preprocessing strategies. We consider parameterized problems, where every instance I comes with a *parameter* k. Such a problem is said to admit a *polynomial kernel* if every instance (I, k) can be reduced in polynomial time to an equivalent instance with both size and parameter value bounded by a polynomial in k. The study of kernelization is a major research frontier of Parameterized Complexity and many important recent advances in the area are on kernelization. These include general results showing that certain classes of parameterized problems have polynomial kernels [3, 10, 22, 26]. The recent development of a framework for ruling out polynomial kernels under certain complexity-theoretic assumptions [9, 17, 23] has added a new dimension to the field and strengthened its connections to classical complexity. For overviews of kernelization we refer to surveys [8, 24] and to the corresponding chapters in books on Parameterized Complexity [21, 30].

While the initial interest in kernelization was driven mainly by practical applications, the notion of kernelization turned out to be very important in theory as well. It is well known, see e.g. [18], that a parameterized problem is fixed parameter tractable, or belongs to the class FPT, if and only if it has a (perhaps exponential) kernel. Kernelization enables us to classify problems within the class FPT further, based on the sizes of the problem kernels. So far, most of the work done in the field of kernelization is still specific to particular problems and powerful unified techniques to identify classes of problems with polynomial kernels are still in their nascent stage. One of the fundamental challenges in the area is the possibility of characterising general classes of parameterized problems possessing kernels of polynomial sizes. From this perspective, the class of the p-$\mathcal{F}$-Deletion problems is very interesting because it contains as special cases the p-Vertex Cover and p-Feedback Vertex Set problems which are the most intensively studied problems from the kernelization perspective.

Our contribution and key ideas. One of the main conceptual contributions of this work is the extension of protrusion techniques, initially developed in [10, 22] for obtaining meta-kernelization theorems for problems on sparse graphs like planar and H-minor-free graphs, to general graphs. We demonstrate this by obtaining a number of kernelization results for the p-$\mathcal{F}$-Deletion problem when $\mathcal{F}$ contains a planar graph. Our first result is the following theorem for graphs that do not contain $K_{1,t}$ (a star on t leaves, see Figure 1).

▶ **Theorem 1.** *Let $\mathcal{F}$ be an obstruction set containing a planar graph. Then p-$\mathcal{F}$-Deletion admits a linear vertex kernel on graphs excluding $K_{1,t}$ as an induced subgraph, where t is a fixed integer.*

Several well studied graph classes do not contain graphs with induced $K_{1,t}$. Of course, every graph with maximum vertex degree at most $(t-1)$ is $K_{1,t}$-free. The class of $K_{1,3}$-free graphs, also known as claw-free graphs, contains line graphs and de Bruijn graphs. Unit disc graphs are known to be $K_{1,7}$-free [15].

Our kernelization is a divide and conquer algorithm which finds large protrusions. A *protrusion* is a subgraph of constant treewidth separated from the remaining part of the graph by a constant number of vertices. Having found protrusions of substantial size, the kernelization algorithm replaces them with smaller, "equivalent" protrusions. Here we use the results from the work by Bodlaender et al. [10] that enable this step whenever the parameterized problem in question "behaves like a regular language". To prove that p-$\mathcal{F}$-Deletion has the desired properties for this step, we formulate the problem in monadic second order logic and show that it exhibits certain monotonicity properties. As a corollary we obtain that p-Feedback Vertex Set, p-Diamond Hitting Set, p-Pathwidth One Deletion Set, and p-Outerplanar Deletion Set admit a linear vertex kernel on graphs excluding $K_{1,t}$ as an induced subgraph. With the same methodology we also obtain a $O(k \log k)$ vertex kernel for p-Disjoint Cycle Packing on graphs excluding $K_{1,t}$ as an induced subgraph. We note that p-Disjoint Cycle Packing does not admit a polynomial kernel on general graphs [11] unless $coNP \subseteq NP/poly$.

Let θ_c be a graph with two vertices and $c \geq 1$ parallel edges (see Figure 1). Our second result is the following theorem on general graphs.

▶ **Theorem 2.** *Let $\mathcal{F}$ be an obstruction set containing only θ_c. Then p-$\mathcal{F}$-Deletion admits a kernel of size $O(k^2 \log^{3/2} k)$.*

A number of well-studied NP-hard combinatorial problems are special cases of p-θ_c-Deletion. When $c = 1$, this is the classical Vertex Cover problem [29]. For $c = 2$, this is another well studied problem, the Feedback Vertex Set problem [4]. When $c = 3$, this is the Diamond Hitting Set problem [20]. Let us note that the size of the best known kernel for $c = 2$ is $O(k^2)$, which is very close to the size of the kernel in Theorem 2. Also, Dell and van Melkebeek proved that no NP-hard vertex deletion problem based on a graph property that is inherited by subgraphs can have kernels of size $O(k^{2-\varepsilon})$ unless $coNP \subseteq NP/poly$ [17] and thus the sizes of the kernels in Theorem 2 are tight up to a polylogarithmic factor.

The proof of Theorem 2 is obtained in a series of non-trivial steps. The very high level idea is to reduce the general case to problem on graphs of bounded degree, which allows us to use the protrusion techniques as in the proof of Theorem 1. However, vertex degree reduction is not straightforward and requires several new ideas. One of the new tools is a generic $O(\log^{3/2} OPT)$-approximation algorithm for the p-$\mathcal{F}$-Deletion problem when the class of excluded minors for $\mathcal{F}$ contains at least one planar graph. More precisely, we obtain the following result, which is of independent interest.

▶ **Theorem 3.** *Let $\mathcal{F}$ be an obstruction set containing a planar graph, and let OPT be the size of the smallest $\mathcal{F}$-hitting set. Given a graph G, in polynomial time we can find a subset $S \subseteq V(G)$ such that $G[V \setminus S]$ contains no element of $\mathcal{F}$ as a minor and $|S| = O(OPT \cdot \log^{3/2} OPT)$.*

While several generic approximation algorithms are known for problems of minimum vertex deletion to obtain subgraphs with property P, like when P is a hereditary property with a finite number of minimal forbidden subgraphs [28], or can be expressed as a universal first order sentence over subsets of edges of the graph [25], we are not aware of any generic approximation algorithm for the case when a property P is characterized by a finite set of forbidden minors.

We then use the approximation algorithm as a subroutine in a polynomial time algorithm that transforms the input instance (G, k) into an equivalent instance (G', k') such that $k' \leq k$ and the maximum degree of G' is bounded by $O(k \log^{3/2} k)$. An important combinatorial tool used in designing this algorithm is the q–Expansion Lemma. For $q = 1$ this lemma is Hall's theorem and its usage is equivalent to the application of the Crown Decomposition technique [1, 14]. After obtaining an equivalent instance with bounded degree, we apply protrusion techniques and prove Theorem 2.

Related work. All non-trivial p-$\mathcal{F}$-Deletion problems are NP-hard [27]. By one of the most well-known consequences of the celebrated Graph Minor theory of Robertson and Seymour the p-$\mathcal{F}$-Deletion problem is non-uniformly fixed parameter tractable. Whenever $\mathcal{F}$ is given explicitly, the problem is uniformly FPT because the excluded minors for the class of graphs that are YES-instances of the p-$\mathcal{F}$-Deletion problem can by computed explicitly [2]. A special case of that problem, when the set $\mathcal{F}$ contains θ_c, has been studied from approximation and parameterized perspectives. In particular, the case of p-θ_1-Deletion or, equivalently, p-Vertex Cover, is the most well-studied problem in Parameterized Complexity. Different kernelization techniques were applied on the problem, eventually resulting in a $2k$-sized vertex kernel [1, 13]. For the kernelization of p-Feedback Vertex Set, or p-θ_2-Deletion, there has been a sequence of dramatic improvements starting from an $O(k^{11})$ vertex kernel by Buragge et al. [12], improved to $O(k^3)$ by Bodlaender [7], and then finally to $O(k^2)$ by Thomassé [34]. Recently Philip et al. [31] and Cygan et al. [16] obtained polynomial kernels for p-Pathwidth One Deletion Set. Constant factor approximation algorithms are known for Vertex Cover and Feedback Vertex Set [4, 5]. Very recently, a constant factor approximation algorithm for the Diamond Hitting Set problem, or p-θ_3-Deletion, was obtained in [20]. Prior to our work, no polynomial kernels were known for p-Diamond Hitting Set or more general families of p-$\mathcal{F}$-Deletion problems.

2 Preliminaries

In this section we give various definitions which we use in the paper. For $n \in \mathbb{N}$, we use $[n]$ to denote the set $\{1, \ldots, n\}$. We use $V(G)$ to denote the vertex set of a graph G, and $E(G)$ to denote the edge set. The degree of a vertex v in G is the number of edges incident on v, and is denoted by $d(v)$. We use $\Delta(G)$ to denote the maximum degree of G. A graph G' is a *subgraph* of G if $V(G') \subseteq V(G)$ and $E(G') \subseteq E(G)$. The subgraph G' is called an *induced subgraph* of G if $E(G') = \{\{u, v\} \in E(G) \mid u, v \in V(G')\}$. Given a subset $S \subseteq V(G)$ the subgraph induced by S is denoted by $G[S]$. The subgraph induced by $V(G) \setminus S$ is denoted by $G \setminus S$. We denote by $N(S)$ the open neighborhood of S, i.e. the set of vertices in $V(G) \setminus S$ adjacent to S. Let $\mathcal{F}$ be a finite set of graphs. A vertex subset $S \subseteq V(G)$ of a graph G is said to be a *$\mathcal{F}$-hitting set* if $G \setminus S$ does not contain any graphs in the family $\mathcal{F}$ as a minor.

By *contracting* an edge (u, v) of a graph G, we mean identifying the vertices u and v, keeping all the parallel edges and removing all the loops. A *minor* of a graph G is a graph H that can be obtained from a subgraph of G by contracting edges. We keep parallel edges after contraction since the graph θ_c which we want to exclude as a minor itself contains parallel edges. Let G, H be two graphs. A subgraph G' of G is said to be a *minor-model* of H in G if G' contains H as a minor. The subgraph G' is a *minimal minor-model* of H in G if no proper subgraph of G' is a minor-model of H in G. A graph class $\mathcal{C}$ is *minor closed* if any minor of any graph in $\mathcal{C}$ is also an element of $\mathcal{C}$. A minor closed graph class $\mathcal{C}$ is *H-minor-free* or simply *H-free* if $H \notin \mathcal{C}$.

Parameterized algorithms and Kernels. A parameterized problem Π is a subset of

$\Gamma^* \times \mathbb{N}$ for some finite alphabet Γ. An instance of a parameterized problem consists of (x,k), where k is called the parameter. A central notion in parameterized complexity is *fixed parameter tractability (FPT)* which means, for a given instance (x,k), solvability in time $f(k)\cdot p(|x|)$, where f is an arbitrary function of k and p is a polynomial in the input size. A kernelization algorithm for a parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that, given $(x,k) \in \Sigma^* \times \mathbb{N}$, outputs, in time polynomial in $(|x|+k)$, a pair $(x',k') \in \Sigma^* \times \mathbb{N}$ such that: (a) $(x,k) \in \Pi$ if and only if $(x',k') \in \Pi$ and (b) $|x'|, k' \leq g(k)$, where g is some computable function. The output instance x' is called the kernel, and the function g is referred to as the size of the kernel. If $g(k) = k^{O(1)}$, then we say that Π admits a polynomial kernel [21].
Tree-width and protrusions. We use standard notions of tree decompositions, tree-width, and nice tree decompositions, which definitions can be found in [10, 22]. Whenever we use nice tree decompositions, we will assume that the root bag is empty. The tree-width of a graph G is denoted by $tw(G)$. Given a graph G and $S \subseteq V(G)$, we define $\partial_G(S)$ as the set of vertices in S that have a neighbor in $V(G) \setminus S$. We say that a set $X \subseteq V(G)$ is an r-protrusion of G if $tw(G[X]) \leq r$ and $|\partial_G(X)| \leq r$. In our paper, we also use concepts of MSO, t-boundaried graphs and their properties, the notion of finite integer index, and strong monotonicity. The definitions of these notions can be found in [10, 22]. Proofs of results labeled with ★ have been omitted due to lack of space.

3 Kernelization for p-$\mathcal{F}$-Deletion on $K_{1,t}$ free graphs

In this section we show that if the obstruction set $\mathcal{F}$ contains a planar graph then the p-$\mathcal{F}$-Deletion problem has a linear vertex kernel on graphs excluding $K_{1,t}$ as an induced subgraph. We start with the following lemma, which is crucial to our kernelization algorithms.

▶ **Lemma 4** (★). *Let $\mathcal{F}$ be an obstruction set containing a planar graph of size h. If G has an $\mathcal{F}$-hitting set S of size at most k, then $\mathbf{tw}(G \setminus S) \leq d$ and $\mathbf{tw}(G) \leq k+d$, where $d = 20^{2(14h-24)^5}$.*

The Protrusion Rule — Reductions Based on Finite Integer Index. We obtain our kernelization algorithm for p-$\mathcal{F}$-Deletion by applying a protrusion based reduction rule. That is, any large r-protrusion for a fixed constant r that depends only on $\mathcal{F}$ (that is, only on the problem) is replaced with a smaller equivalent r-protrusion. For this, we utilize the following lemma of Bodlaender et al. [10].

▶ **Lemma 5** ([10]). *Let Π be a problem that has finite integer index. Then there exists a computable function $\gamma : \mathbb{N} \to \mathbb{N}$ and an algorithm that given an instance (G,k) and an r-protrusion X of G of size at least $\gamma(r)$, runs in $O(|X|)$ time and outputs an instance (G^*,k^*) such that $|V(G^*)| < |V(G)|$, $k^* \leq k$, and $(G^*,k^*) \in \Pi$ if and only if $(G,k) \in \Pi$.*

Remark: Let us remark that if G does not have $K_{1,t}$ as an induced subgraph then the proof of Lemma 5 also ensures that the graph G' does not contain $K_{1,t}$ as an induced subgraph. This ensures that the reduced instance belongs to the same graph class as the original.

In order to apply Lemma 5 we need to be able to efficiently find large r-protrusions whenever the instance considered is large enough. Also, we need to prove that p-$\mathcal{F}$-Deletion has finite integer index. The next lemma yields a divide and conquer algorithm for efficiently finding large r-protrusions.

▶ **Lemma 6.** *There is a linear time algorithm that given an n-vertex graph G and a set $X \subseteq V(G)$ such that $\mathbf{tw}(G \setminus X) \leq d$, outputs a $2(d+1)$-protrusion of G of size at least $\frac{n-|X|}{4|N(X)|+1}$. Here d is some constant.*

Proof. Let $F = G \setminus X$. The algorithm starts by computing a nice tree decomposition of F of width at most d. Notice that since d is a constant this can be done in linear time [6]. Let S be the vertices in $V(F)$ that are neighbors of X in G, that is, $S = N_G(X)$.

The nice tree decomposition of F is a pair $(T, \mathcal{B} = \{B_\ell\}_{\ell \in V(T)})$, where T is a rooted binary tree. We will now *mark* some of the nodes of T. For every $v \in S$, we mark the topmost node ℓ in T such that $v \in B_\ell$. In this manner, at most $|S|$ nodes are marked. Now we mark more nodes of T by exhaustively applying the following rule: if u and v are marked, mark their least common ancestor in T. Let M be the set of all marked nodes of T. Standard counting arguments on trees give that $|M| \leq 2|S|$.

Since T is a binary tree, it follows that $T \setminus M$ has at most $2|M|+1$ connected components. Let the vertex sets of these connected components be $C_1, C_2 \ldots C_\eta$, $\eta \leq 2|M| + 1$. For every $i \leq \eta$, let $C_i' = N_T(C_i) \cup C_i$ and let $P_i = \bigcup_{u \in C_i'} B_u$. By the construction of M, every component of $T \setminus M$ has at most 2 neighbors in M. Also for every $1 \leq i \leq \eta$ and $v \in S$, we have that if $v \in P_i$, then v should be contained in one of the bags of $N_T(C_i)$. In other words, $S \cap P_i \subseteq \bigcup_{u \in C_i' \setminus C_i} B_u$. Thus every P_i is a $2(d+1)$-protrusion of G. Since $\eta \leq 2|M| + 1 \leq 4|S| + 1$, the pigeon-hole principle yields that there is a protrusion P_i with at least $\frac{n-|X|}{4|S|+1}$ vertices. The algorithm constructs M and $P_1 \ldots P_\eta$ and outputs the largest protrusion P_i. It is easy to implement this procedure to run in linear time. ◀

The following lemma follows from [10], where it is shown that every strongly monotone p-MIN-MSO problem has finite integer index.

▶ **Lemma 7 (★).** *p-$\mathcal{F}$-DELETION has finite integer index.*

Analysis and Kernel Size – Proof of Theorem 1. Now we give the desired kernel for p-$\mathcal{F}$-DELETION. We will use the following combinatorial lemma.

▶ **Lemma 8 (★).** *Let G be a graph excluding $K_{1,t}$ as an induced subgraph and S be an $\mathcal{F}$-hitting set. If $\mathcal{F}$ contains a planar graph of size h, then $|N(S)| \leq g(h,t) \cdot |S|$ for some computable function g of h and t.*

Proof of Theorem 1. Let (G, k) be an instance of p-$\mathcal{F}$-DELETION and h be the size of a smallest planar graph in the obstruction set $\mathcal{F}$. We first apply Theorem 3 (to be proved in next section), an approximation algorithm for p-$\mathcal{F}$-DELETION with factor $O(\log^{3/2} OPT)$, and obtain a set X such that $G \setminus X$ contains no graph in $\mathcal{F}$ as a minor. If the size of the set X is more than $O(k \log^{3/2} k)$ then we return that (G, k) is a NO-instance to p-$\mathcal{F}$-DELETION. This is justified by the approximation guarantee provided by the Theorem 3.

Let d denote the treewidth of the graph after the removal of X, that is, $d := \mathbf{tw}(G \setminus S)$. Now we obtain the kernel in two phases: we first apply the protrusion rule selectively (Lemma 5) and get a polynomial kernel. Then, we apply the protrusion rule exhaustively on the obtained kernel to get a smaller kernel. This is done in order to reduce the running time complexity of the kernelization algorithm. To obtain the kernel we follow the following steps.

Applying the Protrusion Rule. By Lemma 4, $d \leq 20^{2(14h-24)^5}$. We apply Lemma 6 and obtain a $2(d+1)$-protrusion Y of G of size at least $\frac{|V(G')|-|X|}{4|N(X)|+1}$. By Lemma 7, p-$\mathcal{F}$-DELETION has finite integer index. Let $\gamma : \mathbb{N} \to \mathbb{N}$ be the function defined in Lemma 5. If $\frac{|V(G')|-|X|}{4|N(X)|+1} \geq \gamma(2d+1)$, then using Lemma 5 we replace the $2(d+1)$-protrusion Y in G and obtain an instance (G^*, k^*) such that $|V(G^*)| < |V(G)|$, $k^* \leq k$, and (G^*, k^*) is a YES-instance of p-$\mathcal{F}$-DELETION if and only if (G, k) is a YES-instance of p-$\mathcal{F}$-DELETION . Recall that G^* also excludes $K_{1,t}$ as an induced subgraph.

Let (G^*, k^*) be a reduced instance with hitting set X. In other words, there is no $(2d+2)$-protrusion of size $\gamma(2d+2)$ in $G^* \setminus X$, and Protrusion Rule no longer applies. We claim that the number of vertices in this graph is bounded by $O(k \log^{3/2} k)$. Indeed, since we cannot apply the Protrusion Rule, we have that $\frac{|V(G^*)|-|X|}{4|N(X)|+1} \leq \gamma(2d+2)$. Because $k^* \leq k$, we have that $|V(G^*)| \leq \gamma(2d+2)(4|N(X)|+1)+|X|$. By Lemma 8, $|N(X)| \leq g(h,d) \cdot |X|$ and thus $|V(G^*)| = O(\gamma(2d+2) \cdot k \log^{3/2} k) = O(k \log^{3/2} k)$. This gives us a polynomial time algorithm that returns a vertex kernel of size $O(k \log^{3/2} k)$.

Now we give a kernel of smaller size. We would like to replace every large $(2d+2)$-protrusion in graph by a smaller one. We find a $(2d+2)$-protrusion Y of size at least $\gamma(2d+2)$ by guessing the boundary $\partial(Y)$ of size at most $2d+2$. This could be performed in time $k^{O(d)}$. So let (G^*, k^*) be the reduced instance on which we cannot apply the Protrusion Rule. If G is a YES-instance then there is a $\mathcal{F}$-hitting set X of size at most k such that $\mathbf{tw}(G \setminus X) \leq d$. Now applying the analysis above with this X yields that $|V(G^*)| = O(k)$. Hence if the number of vertices in the reduced instance G^*, to which we can not apply the Protrusion Rule, is more than $O(k)$ then we return that G is a NO-instance. This concludes the proof of the theorem. ◀

▶ **Corollary 9.** *p-*Feedback Vertex Set*, p-*Diamond Hitting Set*, p-*Pathwidth One Deletion Set*, p-*Outerplanar Deletion Set* admit linear vertex kernel on graphs excluding $K_{1,t}$ as an induced subgraph.*

The methodology used in proving Theorem 1 is not limited to p-$\mathcal{F}$-Deletion. For example, it is possible to obtain an $O(k \log k)$ vertex kernel on $K_{1,t}$-free graphs for p-Disjoint Cycle Packing, which is for a given graph G and positive integer k to determine if there are k vertex disjoint cycles in G. It is iteresting to note that p-Disjoint Cycle Packing does not admit a polynomial kernel on general graphs [11]. For our kernelization algorithm, we use the following Erdős-Pósa property [19]: given a positive integer ℓ every graph G either has ℓ vertex disjoint cycles or there exists a set $S \subseteq V(G)$ of size at most $O(\ell \log \ell)$ such that $G \setminus S$ is a forest. So given a graph G and an integer k, we first apply the factor 2 approximation algorithm given in [4] and obtain a set S such that $G \setminus S$ is a forest. If the size of S is more than $O(k \log k)$ then we return that G has k vertex disjoint cycles. Else, we use the fact that p-Disjoint Cycle Packing [10] has finite integer index and apply the protrusion reduction rule in $G \setminus S$ to obtain an equivalent instance (G^*, k^*), as in Theorem 1. The analysis for kernel size used in the proof of Theorem 1 together with the observation that $\mathbf{tw}(G \setminus S) \leq 1$ shows that if (G,k) is a YES instance then the size of $V(G^*)$ is at most $O(k \log k)$.

▶ **Corollary 10.** *p-*Disjoint Cycle Packing* has $O(k \log k)$ vertex kernel on graphs excluding $K_{1,t}$ as an induced subgraph.*

Next, we extend the methods used in this section for obtaining kernels for p-$\mathcal{F}$-Deletion on graphs excluding $K_{1,t}$ as an induced graph to all graphs, though for restricted $\mathcal{F}$ — we consider the families $\mathcal{F}$ that contain θ_c. However, for this kernelization result, we need a polynomial time approximation algorithm with a factor polynomial in optimum size and not depending on the input size. For example, an approximation algorithm with factor $O(\log n)$ would not serve our purpose. We obtain an approximation algorithm (Theorem 3) for p-$\mathcal{F}$-Deletion with a factor $O(\log^{3/2} OPT)$ whenever the finite obstruction set $\mathcal{F}$ contains a planar graph. Here OPT is the size of a minimum $\mathcal{F}$-hitting set. This immediately implies a factor $O(\log^{3/2} n)$ algorithm for all the problems that can categorized by p-$\mathcal{F}$-Deletion. The proof of Theorem 3 is crucially based on the following lemma.

▶ **Lemma 11 (★).** *There is a polynomial time algorithm that, given a graph G and a positive integer k, either reports that G has no $\mathcal{F}$-hitting set of size at most k or finds an $\mathcal{F}$-hitting set of size at most $O(k \log^{3/2} k)$.*

4 Kernelization for p-θ_c-Deletion

In this section we obtain a polynomial kernel for p-θ_c-Deletion on general graphs. To obtain our kernelization algorithm we not only need the approximation algorithm mentioned before but also a variation of classical Hall's theorem. We first present this combinatorial tool and other auxiliary results that we make use of.

Combinatorial Lemma and some Linear-Time Subroutines. We need a variation of the celebrated Hall's Theorem, which we call the q–Expansion Lemma. It is a generalization of a result due to Thomassé [34, Theorem 2.3], and captures a certain property of neighborhood sets in graphs that implicitly has been used by several authors to obtain polynomial kernels for many graph problems. For $q = 1$, the application of this lemma is exactly the well-known Crown Reduction Rule [1, 14].

The Expansion Lemma. Consider a bipartite graph G with vertex bipartition $A \uplus B$. Given subsets $S \subseteq A$ and $T \subseteq B$, we say that S has $|S|$ q-stars in T if to every $x \in S$ we can associate a subset $F_x \subseteq N(x) \cap T$ such that (a) for all $x \in S$, $|F_x| = q$; (b) for any pair of vertices $x, y \in S$, $F_x \cap F_y = \emptyset$. Observe that if S has $|S|$ q-stars in T then every vertex x in S could be thought of as the center of a star with its q leaves in T, with all these stars being vertex-disjoint. Further, a collection of $|S|$ q-stars is also a family of q edge-disjoint matchings, each saturating S. We use the following result in our kernelization algorithm to bound the degrees of vertices.

▶ **Lemma 12** (★). **[The q–Expansion Lemma]** *Let q be a positive integer, and let m be the size of the maximum matching in a bipartite graph G with vertex bipartition $A \uplus B$. If $|B| > mq$, and there are no isolated vertices in B, then there exist nonempty vertex sets $S \subseteq A, T \subseteq B$ such that S has $|S|$ q-stars in T and no vertex in T has a neighbor outside S. Furthermore, the sets S, T can be found in time polynomial in the size of G.*

▶ Observation 1 (★). For $c \geq 2$, any minimal θ_c minor-model M of a graph G is a connected subgraph of G, and does not contain a vertex whose degree in M is less than 2, or a vertex whose deletion from M results in a disconnected graph (a cut vertex of M).

▶ **Lemma 13** (★). *Let G be a graph and v a vertex of G. Given a tree decomposition of width $t \in O(1)$ of G, in $O(n)$ time we can find both (1) a smallest set $S \subseteq V$ of vertices of G such that the graph $G \setminus S$ does not contain θ_c as a minor, and (2) a largest collection $\{M_1, M_2, \ldots, M_l\}$ of θ_c minor models of G such that for $1 \leq i < j \leq l, (V(M_i) \cap V(M_j)) = \{v\}$.*

Now we describe the reduction rules used by the kernelization algorithm. In contrast to the reduction rules employed by most known kernelization algorithms, these rules cannot always be applied on general graphs in polynomial time. Hence the algorithm does not proceed by applying these rules exhaustively, as is typical in kernelization programs. We describe how to arrive at situations where these rules can in fact be applied in polynomial time, and prove that even this selective application of rules results in a kernel of size polynomial in the parameter k.

Bounding the Maximum Degree of a Graph Now we present a set of reduction rules which, given an input instance (G, k) of p-θ_c-Deletion, obtains an equivalent instance (G', k') where $k' \leq k$ and the maximum degree of G' is at most a polynomial in k. In the sequel a vertex v is *irrelevant* if it is not a part of any θ_c minor model, and is *relevant* otherwise. For each rule below, the input instance is (G, k).

▶ Reduction Rule 1 (Irrelevant Vertex Rule). Delete all irrelevant vertices in G.

Given a graph G and a vertex $v \in V(G)$, an *ℓ-flower passing through v* is a set of ℓ different θ_c minor-models in G, each containing v and no two sharing any vertex other than v.

▶ Reduction Rule 2 (Flower Rule). If a $(k+1)$-flower passes through a vertex v of G, then include v in the solution and remove it from G to obtain the equivalent instance $(G \setminus \{v\}, (k-1))$.

The argument for the soundness of these reduction rules is simple and is hence omitted. One can test whether a particular vertex v is part of any minimal minor-model corresponding to θ_c using the rooted minor testing algorithm of Robertson and Seymour [32]. It is not clear, however, that one might check whether a vertex is a part of $(k+1)$-flower in polynomial time. Hence we defer the application of these rules and apply them only when the vertices are "evidently" irrelevant or finding a flower can be solved in polynomial time. Now we state an auxiliary lemma which will be useful in bounding the maximum degree of the graph.

▶ **Lemma 14** (★). *Let G be a n-vertex graph containing θ_c as a minor and v be a vertex such that $G' = G \setminus \{v\}$ does not contain θ_c as a minor and the maximum size of a flower containing v is at most k. Then there exists a set T_v of size $O(k)$ such that $v \notin T_v$ and $G \setminus T_v$ does not contain θ_c as a minor. Moreover we can find the set T_v in polynomial time.*

▶ **Lemma 15.** *There exists a polynomial time algorithm that, given an instance (G, k) of p-θ_c-Deletion returns an equivalent instance (G', k') such that $k' \leq k$ and that the maximum degree of G' is $O(k \log^{3/2} k)$. Moreover it also returns a θ_c-hitting set of G' of size $O(k \log^{3/2} k)$.*

Proof. Given an instance (G, k) of p-θ_c-Deletion, we first apply Lemma 11 on (G, k). The polynomial time algorithm described in Lemma 11, given a graph G and a positive integer k either reports that G has no θ_c-hitting set of size at most k, or finds a θ_c-hitting set of size at most $k^* = O(k \log^{3/2} k)$. If the algorithm reports that G has has no θ_c-hitting set of size at most k, then we return that (G, k) is a NO-instance to p-θ_c-Deletion. So we assume that we have a hitting set $\mathcal{S}$ of size k^*. Now we proceed with the following two rules.

Selective Flower Rule. To apply the Flower Rule selectively we use $\mathcal{S}$, the θ_c-hitting set. For a vertex $v \in \mathcal{S}$ let $\mathcal{S}_v := \mathcal{S} \setminus \{v\}$ and let $G_v := G \setminus \mathcal{S}_v$. By a result of Robertson et. al. [33] we know that any graph of treewidth greater than 20^{2c^5} contains a $c \times c$ grid, and hence θ_c, as a minor. Since deleting v from G_v makes it θ_c-minor-free, $\mathbf{tw}(G_v) \leq 20^{2c^5} + 1 = O(1)$. Now by Lemma 13, we find in linear time the size of the largest flower centered at v, in G_v. If for any vertex $v \in \mathcal{S}$ the size of the flower in G_v is at least $k+1$, we apply the Flower Rule and get an equivalent instance $(G \leftarrow G \setminus \{v\}, k \leftarrow k-1)$. Furthermore, we set $\mathcal{S} := \mathcal{S} \setminus \{v\}$. We apply the Flower Rule selectively until no longer possible. We abuse notation and continue to use (G, k) to refer to the instance that is reduced with respect to exhaustive application of the Selective Flower Rule. Thus, for every vertex $v \in \mathcal{S}$ the size of any flower passing through v in G_v is at most k.

Now we describe how to find, for a given $v \in V(G)$, a hitting set $H_v \subseteq V(G) \setminus \{v\}$ for all minor-models of θ_c that contain v. Notice that this hitting set is required to *exclude* v, so H_v cannot be the trivial hitting set $\{v\}$. If $v \notin \mathcal{S}$, then $H_v = \mathcal{S}$. On the other hand, suppose $v \in \mathcal{S}$. Since the maximum size of a flower containing v in the graph G_v is at most k, by Lemma 14, we can find a set T_v of size $O(k)$ that does not contain v and hits all the θ_c minor-models passing through v in G_v. Hence in this case we set $H_v = \mathcal{S}_v \cup T_v$. We denote $|H_v|$ by h_v. Notice that H_v is defined algorithmically, that is, there could be many small hitting sets in $V(G) \setminus \{v\}$ hitting all minor-models containing v, and H_v is one of them.

q-expansion Rule with $q = c$. Given an instance (G, k), $\mathcal{S}$, and a family of sets H_v, we show that if there is a vertex v with degree more than $ch_v + c(c-1)h_v$, then we can reduce its degree to at most $ch_v + c(c-1)h_v$ by repeatedly applying the q–Expansion Lemma with $q = c$. Observe that for every vertex v the set H_v is also a θ_c hitting set for G, that is, H_v

hits *all* minor-models of θ_c in G. Consider the graph $G \setminus H_v$. Let the components of this graph that contain a neighbor of v be $C_1, C_2, \ldots, C_r$. Note that v cannot have more than $(c-1)$ neighbors into any component, else contracting the component will form a θ_c minor and will contradict the fact that H_v hits all the θ_c minors. Also note that none of the C_i's can contain a minor model of θ_c.

We say that a component C_i is adjacent to H_v if there exists a vertex $u \in C_i$ and $w \in H_v$ such that $(u, w) \in E(G)$. Next we show that vertices in components that are not adjacent to H_v are irrelevant in G. Recall a vertex is irrelevant if there is no minimal minor model of θ_c that contains it. Consider a vertex u in a component C that is not adjacent to H_v. Since $G[V(C) \cup \{v\}]$ does not contain any θ_c minor we have that if u is a part of a minimal minor model $M \subseteq G$, then $v \in M$ and also there exists a vertex $u' \in M$ such that $u' \notin C \cup \{v\}$. Then the removal of v disconnects u from u' in M, a contradiction to Observation 1 that for $c \geq 2$, any minimal θ_c minor model M of a graph G does not contain a cut vertex. Applying the Irrelevant Vertex Rule to the vertices in all such components leaves us with a new set of components $D_1, D_2, \ldots, D_s$, such that for every i, in D_i, there is at least one vertex that is adjacent to a vertex in H_v.

As before, we continue to use G to refer to the graph obtained after the Irrelevant Vertex Rule has been applied in the context described above. We also update the sets H_v for $v \in V(G)$ by deleting all the vertices w from these sets those have been removed using Irrelevant Vertex Rule.

Now, consider a bipartite graph $\mathcal{G}$ with vertex bipartitions H_v and D. Here $D = \{d_1, \ldots, d_s\}$ contains a vertex d_i corresponding to each component D_i. For every $u \in H_v$, we add the edge (u, d_i) if there is a vertex $w \in D_i$ such that $\{u, w\} \in E(G)$. Even though we start with a simple graph (graphs without parallel edges) it is possible that after applying reduction rules parallel edges may appear. However, throughout the algorithm, we ensure that the number of parallel edges between any pair of vertices is at most c. Now, v has at most ch_v edges to vertices in H_v. Since v has at most $(c-1)$ edges to each D_i, it follows that if $d(v) > ch_v + c(c-1)h_v$, then the number of components $|D|$ is more than ch_v. Now by applying q–Expansion Lemma with $q = c$, $A = H_v$, and $B = D$, we find a subset $S \subseteq H_v$ and $T \subseteq D$ such that S has $|S|$ c-stars in T and $N(T) = S$.

The reduction rule involves deleting edges of the form (v, u) for all $u \in D_i$, such that $d_i \in T$, and adding c edges between v and w for all $w \in S$. We add these edges only if they were not present before so that the number of egdes between any pair of vertices remains at most c. This completes the description of the q-expansion reduction rule with $q = c$. Let G_R be the graph obtained after applying the reduction rule. The following lemma shows the correctness of the rule.

▶ **Lemma 16** (★)**.** *Let G, S and v be as above and G_R be the graph obtained after applying the c-expansion rule. Then (G, k) is a YES instance of p-θ_c-*Deletion *if and only if (G_R, k) is a YES instance of p-θ_c-*Deletion*.*

Observe that all edges that are added during the application of the q-expansion reduction rule have at least one end point in $\mathcal{S}$, and hence $\mathcal{S}$ remains a hitting set of G_R. We are now ready to summarize the algorithm that bounds the degree of the graph (see Algorithm 1).

Let the instance output by Algorithm 1 be $(G', k', \mathcal{S})$. Clearly, in G', the degree of every vertex is at most $ch_v + c(c-1)h_v \leq O(k \log^{3/2} k)$. The routine also returns $\mathcal{S}$ — a θ_c-hitting set of G' of size at most $O(k \log^{3/2} k)$.

We now show that the algorithm runs in polynomial time. For $x \in V(G)$, let $\nu(x)$ be the number of neighbors of x to which x has fewer than c parallel edges. Observe that the

Algorithm 1 BOUND-DEGREE$(G, k, \mathcal{S})$

```
Apply the Selective Flower Rule
if $\exists v \in V(G)$ such that $d(v) > ch_v + c(c-1)h_v$ then
    Apply the $q$-expansion reduction rule with $q = c$.
else
    Return $(G, k, \mathcal{S})$.
end if
Return BOUND-DEGREE$(G, k, \mathcal{S})$.
```

application of q-expansion reduction rule never increases $\nu(x)$ for any vertex and decreases $\nu(x)$ for at least one vertex. The other rules delete vertices, which can never increase $\nu(x)$ for any vertex. This concludes the proof. ◀

5 Conclusion

In this paper we gave the first kernelization algorithms for a subset of p-$\mathcal{F}$-DELETION problems and a generic approximation algorithm for the p-$\mathcal{F}$-DELETION problem when the set of excluded minors $\mathcal{F}$ contains at least one planar graph. Our approach generalizes and unifies known kernelization algorithms for p-VERTEX COVER and p-FEEDBACK VERTEX SET. By the celebrated result of Robertson and Seymour, every p-$\mathcal{F}$-DELETION problem is FPT and our work naturally leads to the following question: does every p-$\mathcal{F}$-DELETION problem have a polynomial kernel? Can it be that for some finite sets of minor obstructions $\mathcal{F} = \{O_1, \dots, O_p\}$ the answer to this question is NO? Even the case $\mathcal{F} = \{K_5, K_{3,3}\}$, vertex deletion to planar graphs, is an interesting challenge. Another interesting question is if our techniques can be extended to another important case when $\mathcal{F}$ contains a planar graph.

References

1 F. N. Abu-Khzam, M. R. Fellows, M. A. Langston, and W. H. Suters. Crown structures for vertex cover kernelization. *Theory Comput. Syst.*, 41(3):411–430, 2007.

2 I. Adler, M. Grohe, and S. Kreutzer. Computing excluded minors. In *SODA*, pages 641–650, 2008.

3 N. Alon, G. Gutin, E. J. Kim, S. Szeider, and A. Yeo. Solving max-r-sat above a tight lower bound. In *SODA*, pages 511–517. ACM-SIAM, 2010.

4 V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM J. Discr. Math.*, 12(3):289–297, 1999.

5 R. Bar-Yehuda and S. Even. A linear-time approximation algorithm for the weighted vertex cover problem. *J. Algorithms*, 2(2):198–203, 1981.

6 H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

7 H. L. Bodlaender. A cubic kernel for feedback vertex set. In *STACS*, volume 4393 of *Lecture Notes in Comput. Sci.*, pages 320–331. Springer, 2007.

8 H. L. Bodlaender. Kernelization: New upper and lower bound techniques. In *IWPEC*, volume 5917 of *Lecture Notes in Computer Science*, pages 17–37. Springer, 2009.

9 H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

10 H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (Meta) Kernelization. In *FOCS*, pages 629–638. IEEE, 2009.

11 H. L. Bodlaender, S. Thomassé, and A. Yeo. Kernel Bounds for Disjoint Cycles and Disjoint Paths. In *ESA*, volume 5757 of *Lecture Notes in Comput. Sci.*, pages 635–646. Springer, 2009.

12 K. Burrage, V. Estivill Castro, M. R. Fellows, M. A. Langston, S. Mac, and F. A. Rosamond. The Undirected Feedback Vertex Set Problem Has a Poly(k) Kernel. In *IWPEC*, volume 4169 of *Lecture Notes in Comput. Sci.*, pages 192–202. Springer, 2006.

13 J. Chen, I. A. Kanj, and W. Jia. Vertex cover: further observations and further improvements. *J. Algorithms*, 41(2):280–301, 2001.

14 B. Chor, M. R. Fellows, and D. W. Juedes. Linear kernels in linear time, or how to save k colors in $O(n^2)$ steps. In *WG*, volume 3353 of *LNCS*, pages 257–269. Springer, 2004.

15 B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Math.*, 86(1-3):165–177, 1990.

16 M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk. Improved fpt algorithm and quadratic kernel for pathwidth one vertex deletion. In *IPEC*, volume 6478 of *Lecture Notes in Comput. Sci.*, pages 95–106. Springer, 2010.

17 H. Dell and D. van Melkebeek. Satisfiability allows no nontrivial sparsification unless the polynomial-time hierarchy collapses. In *STOC*, pages 251–260. ACM, 2010.

18 R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1998.

19 P. Erdős and L. Pósa. On independent circuits contained in a graph. *Canadian J. Math.*, 17:347–352, 1965.

20 S. Fiorini, G. Joret, and U. Pietropaoli. Hitting diamonds and growing cacti. In *IPCO 2010*, volume 6080 of *Lecture Notes in Comput. Sci.*, pages 191–204. Springer, 2010.

21 J. Flum and M. Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

22 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and kernels. In *SODA*, pages 503–510. ACM-SIAM, 2010.

23 L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *STOC*, pages 133–142. ACM, 2008.

24 J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007.

25 P. G. Kolaitis and M. N. Thakur. Approximation properties of NP minimization classes. *J. Comput. System Sci.*, 50:391–411, 1995.

26 S. Kratsch. Polynomial kernelizations for MIN $\mathrm{F}^{+}\mathrm{pi}_1$ and MAX NP. In *STACS*, volume 3 of *(LIPIcs)*, pages 601–612. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2009.

27 J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. of Comp. System Sci.*, 20(2):219 – 230, 1980.

28 C. Lund and M. Yannakakis. The approximation of maximum subgraph problems. In *ICALP*, volume 700 of *Lecture Notes in Comput. Sci.*, pages 40–51. Springer, 1993.

29 G. L. Nemhauser and L. E. Trotter, Jr. Properties of vertex packing and independence system polyhedra. *Math. Programming*, 6:48–61, 1974.

30 R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications.* Oxford University Press, Oxford, 2006.

31 G. Philip, V. Raman, and Y. Villanger. A quartic kernel for pathwidth-one vertex deletion. In *WG*, volume 6410 of *Lecture Notes in Computer Science*, pages 196–207, 2010.

32 N. Robertson and P. D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Comb. Theory Ser. B*, 63:65–110, 1995.

33 N. Robertson, P. D. Seymour, and R. Thomas. Quickly excluding a planar graph. *J. Comb. Theory Ser. B*, 62:323–348, 1994.

34 S. Thomassé. A quadratic kernel for feedback vertex set. *ACM Transactions on Algorithms*, 6(2), 2010.

Self-Assembly of Arbitrary Shapes Using RNAse Enzymes: Meeting the Kolmogorov Bound with Small Scale Factor (extended abstract)*

Erik D. Demaine[1], Matthew J. Patitz[2], Robert T. Schweller[3], and Scott M. Summers[4]

1 MIT Computer Science and Artificial Intelligence Laboratory,
32 Vassar St., Cambridge, MA 02139, USA.
edemaine@mit.edu

2 Department of Computer Science, University of Texas–Pan American,
Edinburg, TX 78539, USA.
mpatitz@cs.panam.edu

3 Department of Computer Science, University of Texas–Pan American,
Edinburg, TX 78539, USA.
schwellerr@cs.panam.edu

4 Department of Computer Science and Software Engineering, University of Wisconsin–Platteville, Platteville, WI 53818, USA.
summerss@uwplatt.edu

Abstract

We consider a model of algorithmic self-assembly of geometric shapes out of square Wang tiles studied in SODA 2010, in which there are two types of tiles (e.g., constructed out of DNA and RNA material) and one operation that destroys all tiles of a particular type (e.g., an RNAse enzyme destroys all RNA tiles). We show that a single use of this destruction operation enables much more efficient construction of arbitrary shapes. In particular, an arbitrary shape can be constructed using an asymptotically optimal number of distinct tile types (related to the shape's Kolmogorov complexity), after scaling the shape by only a logarithmic factor. By contrast, without the destruction operation, the best such result has a scale factor at least linear in the size of the shape and is connected only by a spanning tree of the scaled tiles. We also characterize a large collection of shapes that can be constructed efficiently without any scaling.

1998 ACM Subject Classification F. Theory of Computation

Keywords and phrases Biomolecular computation, RNAse enzyme self-assembly, algorithmic self-assembly, Komogorov complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.201

1 Introduction

DNA self-assembly research attempts to harness the power of synthetic biology to manipulate matter at the nanoscale. The general goal of this field is to design a simple system of particles (e.g., DNA strands) that efficiently assemble into a desired macroscale object. Such technology is fundamental to the field of nanotechnology and has the potential to allow for massively parallel, bottom-up fabrication of complex nanodevices, or the implementation of

* This research was supported in part by NSF grant CDI-0941538.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 201–212
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

a biological computer. Motivated by experimental DNA assemblies of basic building blocks or DNA *tiles* [5, 7, 14, 16, 17, 19, 26], the *tile self-assembly model* [18] has emerged as a premier theoretical model of self-assembly. Tile self-assembly models particles as four-sided Wang tiles which float randomly in the plane and stick to one another when abutting edges have sufficient affinity for attachment.

Perhaps the most fundamental question within the tile self-assembly model is how efficiently, in terms of the number of distinct tile types needed, can a target shape be uniquely assembled. For some special classes of shapes such as rectangles and squares, the problem has been considered in depth under a number of tile-based self-assembly models. More generally, researchers have considered the complexity of assembling arbitrary shapes [9, 11, 20]. In particular, Soloveichik and Winfree [20] show that any shape, modulo scaling, can be self-assembled with a number of tile types close to the Kolmogorov complexity of the target shape. While intriguing from a theoretical standpoint, this result has an important drawback: it assembles an arbitrarily large scaled-up version of the target shape, rather than the exact target shape. It is conceivable that a reasonable scale factor could be tolerated in practice by simply engineering smaller tiles, but the scale factors needed for the Soloveichik-Winfree construction are unbounded in general, proportional to the running time of the Kolmogorov machine that generates the shape, which is at least linear in the size of the target shape in all cases. This extreme resolution loss motivates the search for a practical model and construction that can achieve extremely small scale factors while retaining the Kolmogorov-efficient tile complexity for general shapes.

Our results. We achieve Kolmogorov-efficient tile complexity of general shapes with a logarithmic bounded scale factor, using the experimentally motivated *Staged RNA Assembly Model (SRAM)* introduced in [1]. The SRAM extends the standard tile self-assembly model by distinguishing all tile types as consisting of either DNA or RNA material. Further, in a second stage of assembly, an *RNase enzyme* may be added to the system which dissolves all RNA tiles, thus potentially breaking assemblies apart and allowing for new assemblies to form. While this modification to the model is simple and practically motivated (the idea was first mentioned in [18]), we show that the achievable scale factor for Kolmogorov-efficient assembly of general shapes drops dramatically: for arbitrary shapes of size n, a scale factor of $O(\log n)$ is achieved, and for a large class of "nice" shapes, the Kolmogorov optimal tile complexity can be achieved without scaling (scale factor 1). Refer to Figure 1. Note that the lower bound proof of [20] holds with a simple modification to the program that simulates self-assembly in the SRAM. Further, we show that arbitrarily large portions of infinite computable patterns of the plane can be weakly assembled within the SRAM. Such assembly has been proved impossible in the standard tile assembly model [13], illustrating an important distinction in the power of SRAM compared to the standard tile assembly model.

In addition to tile complexity and scale factor, we also address the metrics of *connectivity* and *addressability*. *Full connectivity* denotes whether all adjacent tiles making up the target shape share positive strength bonds, a desirable property as it creates a stable final assembly. All of our finite constructions are fully connected, unlike the previous result of [20] which just connected a spanning tree of the scaled tiles, making for a potentially very floppy construction. *Addressability* denotes whether a construction is able to assign arbitrary binary labels to the tiles that make up the final assembly. Addressability may have important practical applications for assemblies that are to serve as scaffolding for the fabrication of nanodevices such as circuits in which specific components must be attached to specific locations in the assembled shape. Our $O(\log n)$-scale construction provides the flexibility to encode an arbitrary binary label within the tile types of each scaled-up position in the

General shape S with n points	**Tile Types**	**Stages**	**Scale**	**Connectivity**
Previous work [20]	$\Theta(K(S)/\log K(S))$	1	unbounded	partial
Arbitrary shapes (Thm. 3.2)	$\Theta(K(S)/\log K(S))$	2	$O(\log n)$	full
"Nice" shapes (Thm. 4.2)	$\Theta(K(S)/\log K(S))$	2	1	full
Infinite computable pattern S	**Tile Types**	**Stages**	**Scale**	**Connectivity**
Computable patterns (Sec. 4.4)	$\Theta(K(S)/\log K(S))$	2	1	partial

Table 1 Summary of the tile complexities, stage complexities, scale factors, and connectivity of our RNA staged assembly constructions compared with relevant previous work. The value $K(S)$ denotes the Kolmogorov complexity of a given shape or pattern S, and n denotes the size of (number of points in) S.

assembled shape, thus yielding a high degree of addressability, while our 1-scale construction allows complete addressability. See [10] for a version of this paper that includes color images and a full technical appendix.

2 Preliminaries

We work in the 2-dimensional discrete space $\mathbb{Z}^2$. Let $U_2 = \{(0,1),(1,0),(0,-1),(-1,0)\}$ be the set of all unit vectors in $\mathbb{Z}^2$. We write $[X]^2$ for the set of all 2-element subsets of a set X. All *graphs* here are undirected graphs, i.e., ordered pairs $G = (V, E)$, where V is the set of *vertices* and $E \subseteq [V]^2$ is the set of *edges*. A *grid graph* is a graph $G = (V, E)$ in which $V \subseteq \mathbb{Z}^2$ and every edge $\{\vec{a}, \vec{b}\} \in E$ has the property that $\vec{a} - \vec{b} \in U_2$. The *full grid graph* on a set $V \subseteq \mathbb{Z}^2$ is the graph $G^{\#}_V = (V, E)$ in which E contains *every* $\{\vec{a}, \vec{b}\} \in [V]^2$ such that $\vec{a} - \vec{b} \in U_2$.

A *shape* is a set $S \subseteq \mathbb{Z}^2$ such that $G^{\#}_S$ is connected. In this paper, we consider scaled-up versions of finite shapes. Formally, if X is a shape and $c \in \mathbb{N}$, then a *c-scaling* of S is defined as the set $S^c = \left\{(x,y) \in \mathbb{Z}^2 \;\middle|\; \left(\left\lfloor \frac{x}{c} \right\rfloor, \left\lfloor \frac{y}{c} \right\rfloor\right) \in X\right\}$. Intuitively, S^c is the shape obtained by replacing each point in S with a $c \times c$ block of points. We refer to the natural number c as the *scaling factor* or *resolution loss*. Note that scaled shapes have been studied extensively in the context of a variety of self-assembly systems [6,9,11,20,25].

Fix some universal Turing machine U. The *Kolmogorov complexity* of a shape S, denoted by $K(S)$, is the size of the smallest program π that outputs an encoding of a list of all the points in S. In other words $K(S) = \min\{|\pi| \mid U(\pi) = \langle S \rangle\}$. The reader is encouraged to consult [21] for a more detailed discussion of Kolmogorov complexity.

Here we give a sketch of a variant of Erik Winfree's abstract Tile Assembly Model (aTAM) [22,23] known as the *two-handed* aTAM, which has been studied previously under various names [2,4,8,9,15,24]. Please see [12] for a more detailed description of the model and our notation.

A *tile type* is a unit square with four sides, each having a *glue* consisting of a *label* (a finite string) and *strength* (0, 1, or 2). We assume a finite set T of tile types, but an infinite number of copies of each tile type, each copy referred to as a *tile.* A *supertile* (a.k.a., *assembly*) is a positioning of tiles on the integer lattice $\mathbb{Z}^2$. Two adjacent tiles in a supertile *interact* if the glues on their abutting sides are equal. Each supertile induces a *binding graph*, a grid graph whose vertices are tiles, with an edge between two tiles if they interact. The supertile is *τ-stable* if every cut of its binding graph has strength at least τ, where the weight of an edge is the strength of the glue it represents. That is, the supertile is stable if at least energy τ is required to separate the supertile into two parts. A *tile assembly system* (TAS) is a pair $\mathcal{T} = (T, \tau)$, where T is a finite tile set and τ is the *temperature*, usually 1 or 2. Throughout this paper $\tau = 2$ (unless explicitly stated otherwise). Given a TAS $\mathcal{T} = (T, \tau)$, a supertile is *producible* if either it is a single tile from T, or it is the τ-stable result of translating two producible assemblies. A supertile α is *terminal* if for every producible supertile β, α and β cannot be τ-stably attached. A TAS is *directed* (a.k.a., *deterministic*, *confluent*) if it has only one terminal, producible supertile. Given a connected shape $X \subseteq \mathbb{Z}^2$, a TAS $\mathcal{T}$ *produces X uniquely* if every producible, terminal supertile places tiles only on positions in X (appropriately translated if necessary).

RNA tiles and RNAse enzyme

In this paper, we assume that each tile type is defined as being composed of either DNA or RNA. By careful selection of the actual nucleotides used to create the glues, tile types of any combination of compositions can bind together. The utility of distinguishing RNA-based tile types comes from that fact that, at prescribed points during the assembly process, the experimenter can add an RNAse enzyme to the solution which causes all tiles composed of RNA to dissolve. We assume that, when this occurs, all portions of all RNA tiles are completely dissolved, including glue portions that may be bound to DNA tiles, returning the previously bound edges of those DNA tiles to unbound states.

More formally, for a given supertile Γ that is stable at temperature τ, when the RNAse enzyme is added, all positions in Γ which are occupied by RNA tiles change to the empty tile. The resultant supertile may not be τ-stable and thus defines a multiset of subsupertiles consisting of the maximal stable supertiles of Γ at temperature τ, denoted by $BREAK_\tau(\Gamma)$.

The plausibility of this model was mentioned by Rothemund and Winfree in [18], and formalized in SODA 2010 [1] when it was combined with the idea of staged assembly [9].

Staged assembly with RNA removals

Staged assembly consists of a finite sequence of stages, modeling the actions taken by an experimenter (e.g., bioengineer). A stage assembly system specifies each stage as either a tile addition stage, in which new tile types are added to the system, or an enzyme stage, in which assembled supertiles are broken into pieces by deleting all occurrences of RNA tile types. In both cases, each stage consists of an initial set of preassembled supertiles from the previous stage, unioned with a new set of tile types in the case of a tile addition stage, or the current supertile set broken into subsupertiles (which may then be able to bind to each other) in the case of an enzyme stage. From this initial set, the output of the stage is determined by the two-handed assembly model, and the stage ends once all supertiles are terminal, meaning that no further bindings can occur. It is only at this point that the next stage can be initiated.

Complexity Measures of Tile Assembly Systems

In this paper, we are primarily concerned with measuring the "complexity" of a tile assembly system with respect to the following metrics. **Tile Complexity**: we say that the *tile complexity* (sometimes called the *program-size complexity* [18]) is the number of unique tile types of the system. **Stage Complexity**: we say that the *stage complexity* is the number of stages that a particular tile system must progress through in order to produce a terminal assembly. (We sometimes also mention the *BREAK* complexity [1], which is simply the number of *BREAK* stages.) **Scale Factor**: we say that a tile system produces a shape S with *scale factor* $c \in \mathbb{N}$ if the system uniquely produces S^c. **Connectivity**: when a tile system produces a terminal assembly in which not every adjacent edge interacts with positive strength, then we say that the system has *partial connectivity.* On the other hand, a tile assembly system achieves *full connectivity* if it only produces terminal assemblies in which every abutting edge interacts with positive strength. **Addressability**: addressability (of the final assembly of a tile assembly system) concerns the ability of a tile system to *address* or mark each tile in the final assembly with a character drawn from $\Sigma = \{0, 1\}$. Note that addressability concerns the ability of tile systems to label certain (tiles placed at) locations in their final assembly as "black" or "nonblack."

3 The Pod Construction

3.1 Partial Connectivity Construction

As a warmup to our main result, we obtain partial connectivity:

▶ **Theorem 3.1.** *For every finite shape $S \subset \mathbb{Z}^2$, there exists a staged RNA assembly system $\mathcal{T}_S$ that uniquely produces S and moreover, $\mathcal{T}_S$ has tile complexity $O\left(\frac{K(S)}{\log K(S)}\right)$, stage complexity 2, a scale factor of $O(\log |S|)$, and has partial connectivity.*

One highlight of Theorem 3.1 is that the stage complexity of $\mathcal{T}_S$ is 2, i.e., the stages in our construction consist of the initial tile addition stage followed by a single *BREAK* stage. This is the fewest stages possible in any construction that makes use of the power of the RNAse enzyme. The remainder of this section is devoted to providing a proof sketch of Theorem 3.1.

At a high level, the construction for Theorem 3.1 works by forming a $O(\log |S|) \times O(\log |S|)$ (roughly) square block to represent each point in S. The correct positioning of blocks is ensured by encoding binary strings that are unique to each pair of adjacent edges as "teeth" on the edges of the blocks. The assembly begins with a seed, composed of RNA tile types representing a Turing machine that outputs S as a list of points. An assembly which simulates that Turing machine and then outputs definitions for each of the blocks assembles first, with all tiles being composed of RNA except for those forming the blocks, which are composed of DNA. We think of these blocks as DNA "pods" growing off of the RNA assembly. A *BREAK* operation is then performed which dissolves everything except for the DNA blocks. These blocks then combine to form the scaled version of S. Details of this construction follow. Figure 1a shows the basic design of the blocks used in this construction. Figure 1b depicts the high level structure of this construction.

The seed row consists of a row of tiles that uniquely self-assemble into a binary representation of the shortest Turing machine M that outputs the definition of a desired shape S as a list of points, and then halts. Note that we use the optimal encoding scheme of [3, 20], which implies that the tile complexity of our construction is $O\left(\frac{K(S)}{\log K(S)}\right)$.

Assembly begins with the "unpacking" phase (similar to the main construction of Soloveichik and Winfree [20]). Once this simulation completes, the top row of the assembly will consist of the list of points in the shape. Next, another Turing machine, N (charged with the task of executing the algorithm defined in Section A.4 of [10]), is simulated by the assembly.

Once N halts, the top row of the assembly will consist of a sequence of binary strings that represent the binary values to be encoded along the edges of the DNA blocks. It is these blocks that will come together in a 2-handed fashion to form the final, scaled version of S. The correct positioning of the blocks is ensured by the patterns of binary teeth as well as the glues on the corners of the blocks which ensure that only complementary corners of blocks can bind (e.g., the northeast corner of one block could bind only to the northwest corner of another). For block edges which correspond to an outer edge of the shape S, instead of binary teeth a smooth edge with 0-strength glues will be formed. Note that the seed tiles, Turing machine simulation tiles, and tiles outside of the blocks are all RNA tile types which will ultimately be dissolved by RNase enzyme in the *BREAK* stage. Following the

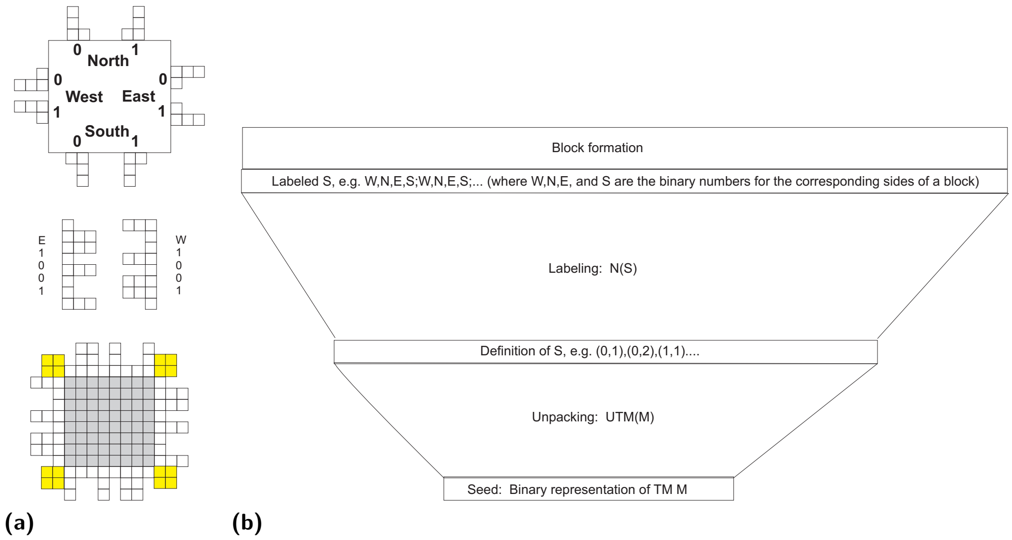

0
North
1
0
West
East
0
1
1
0
South
1
E
1
0
0
1
W
1
0
0
1
Block formation
Labeled S, e.g. W,N,E,S;W,N,E,S;... (where W,N,E, and S are the binary numbers for the corresponding sides of a block)
Labeling: N(S)
Definition of S, e.g. (0,1),(0,2),(1,1)....
Unpacking: UTM(M)
Seed: Binary representation of TM M
(a)
(b)

Figure 1 (a) Top: Key showing the shapes assembled for bits on each side of a block. Middle: Example East side and West side, each representing the bit pattern "1001". Bottom: Example block which has the bit pattern "1001" on each side. Note that the white tiles represent the binary patterns and have null glues on their outer edges while each exposed side of each yellow block has a single strength 1 glue exposed which is specific to its corner and direction. (b) High level overview of the main components of the pod construction.

BREAK, the $O(\log|S|) \times O(\log|S|)$ sized blocks representing each of the points in S are free to self-assemble into the scaled up version of S, thus completing the construction.

3.2 Full Addressability of Points in S

In the aTAM, tile types are allowed to have "labels" which are nonfunctional (not necessarily unique) strings associated with each tile type. Often, labels are assigned to tile types to make it easier to logically identify and group them (for instance, the "0" and "1" labels assigned to the tile types that assemble into a binary counter). In laboratory implementations of DNA tile types, tile types are often created with the equivalent of such binary labels by the inclusion or exclusion of a hairpin loop structure which projects upward above the plane of the tile, for 0 and 1 respectively (a notable example of this technique is due to Papadakis, Rothemund and Winfree [19]). This is currently done to simplify the imaging process and therefore the detection of errors that occur in the assembly. However, it is possible that in the future such projecting labels could be also used to create binding sites for additional materials, allowing the self-assembling structure to serve as a scaffolding for more complicated productions. For simplicity, we let the set of available labels be $\Sigma = \{0, 1\}$.

Here we present a construction that facilitates the arbitrary assignment of labels to subsets of locations in the final assembly. We consider such locations to be "addressable." This provides a method for associating labels, in the form of binary strings, with each of the points in S. These binary strings will be represented by rows of tiles within the blocks, each labeled with a "0" or "1."

In the construction for Theorem 3.1, it is trivial to allow the TM M encoded in the seed to also output a binary string to be used to label each/any point in S. This binary string can be passed upward through the south sides of the DNA blocks so that they are represented by the labels of the tile types which form the center of each block (either in particular, designated rows or in all rows). Of course, doing so requires an appropriate increase in tile complexity—the *additional* complexity of encoding each string that will ultimately be printed on (e.g., used to address) each supertile in the final assembly.

This labeling method allows bit strings of length at most the width of the center portion of a DNA block (plus 2 additional tiles) to be specified for each DNA block. Only one such unique label can be specified for each block, but the row (or rows) in which it appears can be specified by M. The label can appear in any subset of the rows, or alternatively in columns. Intuitively, this is done by including a label value, which passes either upward or to the right as the center of the block assembles. At rows (or columns) that have been specified with special markers as M output the definition of the block, the label values can be "expressed" by tile types with the labels corresponding to the bit values.

3.3 Full Connectivity Construction

Recall that for the previous constructions, the only positive strength interaction between the glues of adjacent blocks occurred at the corners of those blocks. We now strengthen Theorem 3.1 as follows.

▶ **Theorem 3.2.** *For every finite shape $S \subset \mathbb{Z}^2$, there exists a staged RNA assembly system $\mathcal{T}_S$ that uniquely produces X and moreover, $\mathcal{T}_S$ has tile complexity $O\left(\frac{K(S)}{\log K(S)}\right)$, stage complexity 2, a scale factor of $O(\log |S|)$, and achieves full connectivity of the terminal assembly.*

A proof sketch of Theorem 3.2 follows. In order to generate shapes with full connectivity, the scheme proposed below requires that the scaling factor be doubled from the construction of Theorem 3.1 and also that, when the RNase enzyme is added, *there are no remaining singleton tiles (neither DNA nor RNA) in the solution*, only the terminally produced assemblies. The latter requirement is due to the fact that the teeth of the blocks produced have single strength glues all along their edges to which single tiles of the correct types could attach and prevent the proper connection of blocks. However, it is easy to remove this assumption by doubling the system temperature from $\tau = 2$ to $\tau = 4$, *and* doubling the strength of every glue that is *internal* to each DNA block while maintaining single strength glues that are on the outside of the block. Note that this additional assumption is not needed for the construction for Theorem 3.1 since with those blocks, there are no locations on the exposed sides to which singleton tiles could attach, only the correct and fully formed complementary blocks.

Figure 2 shows the procedure by which the values for the edges of a block are moved into the necessary positions relative to the edges of the block to be formed. It also shows how those values are turned into "casts" formed of RNA tiles. The high level idea is that first, before any DNA tiles can attach to the assembly, RNA tiles form a "cast" whose shape is the complement of the teeth of the block. Once the self-assembly of the portion of the cast for an edge is completed, the assembly of the DNA teeth for that side is allowed to proceed. The cast is formed as a one-tile-wide path of tiles whose order of growth is generally clockwise. Once the self-assembly of the entire cast is completed, the DNA tiles can fully form the block. Every DNA tile has strength-1 glues on every edge and attaches with its south and west sides as input sides, generally forming the block from the bottom left to the top right (more details of the cast formation can be found

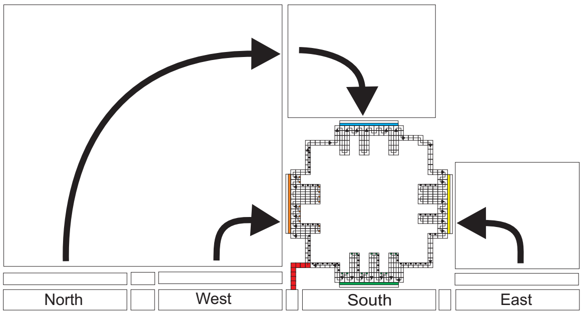


Figure 2 Positioning of block edge information.

in Section A.7 of [10]). Once the blocks form, the remainder of the construction proceeds similarly to the prior construction.

4 Self-Assembly of Shapes without Scaling

4.1 A Bounded Rectangle Decomposition of an Arbitrary Shape

We will now show how the pod construction of Section 3 can be modified to reduce the scale factor from $O(\log |S|)$ to 1 for a large class of finite shapes, while still obtaining asymptotically optimal tile complexity (according to the Kolmogorov complexity of the target shape), using just a single *BREAK* stage, and maintaining full connectivity of the final assembly. The large class of shapes will be the set of shapes that have a "bounded rectangle decomposition"–the definition of which follows.

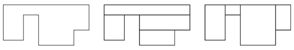


Figure 3 A shape to be formed (left) and the possible rectangle decompositions thereof (middle,right).

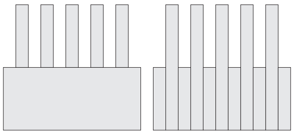


Figure 4 One possible decomposition (among several) of a shape into rectangles.

The leftmost image in Figure 3 shows an example of a simple target shape to be assembled. The middle and rightmost images show two different possible rectangle decompositions of that shape. Instead of having binary teeth along the full edges of each constituent rectangle, binary teeth need only be present at the locations where rectangles must come together, i.e., at the *interface* between two rectangles. The remainder of the outside edges can be made smooth, with 0-strength glues. Throughout this section, S denotes an arbitrary finite shape.

A shape R is a rectangle if $R = \{(x, y) \in \mathbb{Z}^2 \mid a \leq x < m + a \text{ and } b \leq y < n + b \text{ for some } a, b, m, n \in \mathbb{N}\}$. In this case, we say that R is a rectangle of width m and height n positioned at (a, b). We say that $\mathcal{R}(S) = \{R_i\}_{i=0}^{k}$, for some $k \in \mathbb{N}$ is a *rectangle decomposition* of S if for all $0 \leq i < k$, R_i is a non-empty rectangle, $\bigcup_{i=0}^{k-1} R_i = S$ and for all $i, j \in \mathbb{N}$ such that $i \neq j$, $R_i \cap R_j = \varnothing$. See Figure 3 for examples. Let $\mathcal{R} = \{R_i\}_{i=0}^{k-1}$ be a rectangle decomposition of S and suppose that R_i and R_j are rectangles in $\mathcal{R}$. For each $\vec{u} \in U_2 = \{(0, 1), (1, 0), (0, -1), (-1, 0)\}$, denote as $I^{\vec{u}}(R_i, R_j)$ the *interface between rectangles R_i and R_j in direction $\vec{u}$*, i.e., $I^{\vec{u}}(R_i, R_j)$ is the set of all points $(x, y) \in R_j$ such that $(x, y) = (w, z) + \vec{u}$ for some $(w, z) \in R_i$. It is easy to see that, for any rectangle decomposition $\mathcal{R}$, $I^{\vec{u}}(R_i, R_j)$ is the unique interface in direction $\vec{u}$ between R_i and R_j *or* $I^{\vec{u}}(R_i, R_j) = \varnothing$. For each $\vec{u} \in U_2$, the *length* of an interface $I^{\vec{u}}(R_i, R_j)$ is $|I(R_i, R_j)|$. For each $\vec{u} \in U_2$, we say that the *orientation of an interface* $I^{\vec{u}}(R_i, R_j)$ is *horizontal* if $\vec{u} \in \{(1, 0), (-1, 0)\}$ and *vertical* if $\vec{u} \in \{(0, 1), (0, -1)\}$. We say that R_i and R_j are *adjacent* if $I^{\vec{u}}(R_i, R_j) \neq \varnothing$ for some $\vec{u} \in U_2$.

▶ **Definition 4.1.** Let $\mathcal{R} = \{R_i\}_{i=0}^{k-1}$ be a rectangle decomposition of S. We say that $\mathcal{R}$ is a *bounded rectangle decomposition* if: (1) for each $l \in \mathbb{N}$,

$$\left|\left\{ \left|I^{\vec{u}}(R_i, R_j)\right| = l \;\middle|\; \vec{u} \in U_2, i, j \in \mathbb{N} \text{ and } R_i, R_j \in \mathcal{R}\right\}\right| \leq 2^{\left\lfloor \frac{l-12}{4} \right\rfloor}$$

and (2) for all $R_i, R_j \in \mathcal{R}$, if R_i and R_j are adjacent (in some particular direction $\vec{u} \in U_2$), then $\left|I^{\vec{u}}(R_i, R_j)\right| \geq 16$.

Definition 4.1 is motivated by the way we will ultimately construct tile interfaces between DNA supertiles in our forth-coming construction (discussed in the next subsection): each supertile-supertile interface of length l can play host to at most $\left\lfloor \frac{l-12}{4} \right\rfloor$ binary "teeth" since we will use 6 tiles for each corner piece and 4 tiles for the representation of each bit in the interface. Intuitively, the first condition in Definition 4.1 says that there cannot be "too many" (i.e., roughly exponentially-many) interfaces of each length in $\mathcal{R}$, whereas the second condition is merely saying that every non-empty interface must be at least a certain length. In order to bypass the limitation imposed by the first condition, the shape could simply be scaled, with a worst possible case being a scale factor of $O(\log |S|)$.

4.2 Self-Assembly of Rectangles of Arbitrary Dimension

The construction for Theorem 3.2 can be modified to prove the following result.

▶ **Theorem 4.2.** *For every finite shape $S \subset \mathbb{Z}^2$, if S has a bounded rectangle decomposition, then there exists a staged RNA assembly system $\mathcal{T}_S$ that uniquely produces S, $\mathcal{T}_S$ has tile complexity $O\left(\frac{K(S)}{\log K(S)}\right)$, utilizes 2 stages with a single BREAK step and achieves full connectivity of the terminal assembly.*

4.3 Full Addressability of Every Tile in the Final Assembly

In this section, we sketch a construction utilizing a single *BREAK* step that assembles shapes (that can be "nicely" decomposed into rectangles) with no scaling, full connectivity, and full addressability (in the form of specifying either a 0 or 1 label to appear in every single tile position of the final assembly). This strengthens Theorem 4.2 with respect to addressability but with an additional increase in tile complexity of $O(K(B))$, where $B \subseteq S$ is the set of points to be addressed, i.e., the set of points in the final assembly at which tiles labeled with a "1" are placed, as well as requiring an additional constraint on the rectangles contained within the rectangle decomposition. For this construction, we require that there is some constant $k \in \mathbb{Z}^+$ that bounds at least one dimension of every rectangle in every valid rectangle decomposition. That is, every rectangle, although potentially arbitrarily long (or wide) in one dimension, must be no longer or wider *in the other dimension*, than k tiles.

The details of how the rectangular blocks for this construction are formed are depicted in Figure 5. Our construction can be thought of to proceed in four logical phases: the unpacking process, self-assembly of the RNA cast, self-assembly of the rectangular supertiles, and self-assembly of the target shape. The main difference with the previous construction is in the complexity of the cast and the order of assembly of the tiles forming the rectangular supertiles. At a high level, this is due to the fact that information about the specific labels, and therefore tile types–that need to eventually occupy every single position–must be propagated from the casts into the forming rectangular supertiles. This forces the constraint on one dimension of each rectangle, and the fact that the construction retains full connectivity forces the positioning of the glues on the cast that propagate the information to be greatly complicated. Details of this construction can be found in Section A.8 of [10], and a high level schematic can be seen in Figure 6.

4.4 Weak Self-Assembly of Computable Patterns

Weak self-assembly is a general notion of self-assembly that applies to the self-assembly of patterns that are in some sense "painted" on a canvas of tiles that strictly contains S (as opposed to *strict self-assembly*, which pertains to the self-assembly of a given target shape and nothing else). Intuitively, we say that a pattern $S \subseteq \mathbb{Z}^2$ weakly self-assembles if there

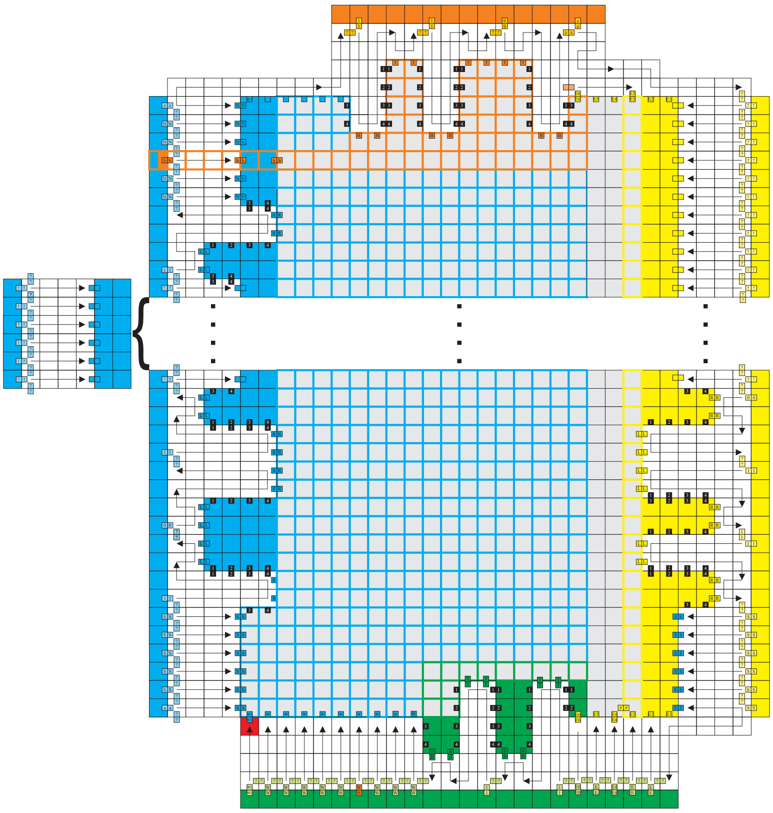


Figure 5 Schematic of the self-assembly of a fixed-width, fully addressable rectangle that will ultimately (after the RNAse enzyme is applied) participate in the self-assembly of a fully connected and fully addressable unique terminal assembly (see Section A.8 of [10] for more details). The cast forms as a single path around the entire perimeter, beginning at the bottom left side. Shaded/colored tiles are DNA tiles while white tiles are RNA. Only colored, non-grey tiles are allowed to assemble before the entire cast assembles. We depict single strength bonds as little colored (and labeled) squares along the edges of tiles. Arrows represent double strength bonds between contiguous groups of tiles through which they pass.

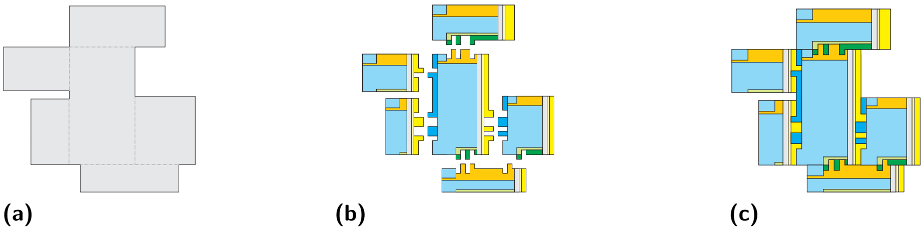


Figure 6 Each individual supertile is colored so as to correspond to the more detailed Figure 5. Note that the bottommost supertile attaches via two north-facing interfaces! (a) An example target shape X and a candidate bounded-rectangle decomposition (b) The corresponding supertiles. For the sake of example, assume the width of each supertile must not exceed that of the bottommost supertile (c) The final assembly

is a tile system that places special "black" marker tiles on—and only on—every point that belongs to the set S.

Our final construction self-assembles an arbitrarily "large" (square) portion of any computable pattern, with the size of the portion of the pattern determined simply by how long the self-assembly is allowed to proceed before the *BREAK* operation is performed. This clearly demonstrates the fact that staged self-assembly with DNA removals is strictly more powerful than the aTAM, in terms of the weak self-assembly of patterns, as it was shown in [13] that there are (decidable) patterns that cannot weakly self-assembles in the aTAM.

Essentially, this assembly simulates a Turing machine using RNA tiles and creates pods for DNA tile rectangles with constant width (or height) and increasingly large height (or width). Tiles on these pods are labeled corresponding to the portion of the pattern which they will occupy. Figure 7 demonstrates the manner in which these rectangles will ultimately combine.

The darker grey portions represent the binary teeth used to connect the rectangles. Note that these rectangle-rectangle interfaces get larger as the rectangles grow out from the center, but since there remain unconnected portions of the perimeters of each rectangle, the final assembly is not fully connected. For infinite patterns, the portion of the construction that performs the Turing machine computation and outputs the definitions of the rectangles must be slightly modified so that the rectangles are formed on the left side of the north-growing simulation, enumerated one after another. This allows for an arbitrarily large portion of such a pattern to be weakly self-assembled by simply allowing the assembly to proceed for a "long enough" period of time before performing the *BREAK* operation.

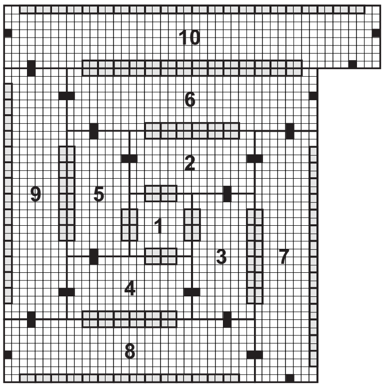


Figure 7 The first 10 rectangles weakly self-assembling an arbitrary computable pattern.

References

1 Zachary Abel, Nadia Benbernou, Mirela Damian, Erik D. Demaine, Martin L. Demaine, Robin Flatland, Scott D. Kominers, and Robert T. Schweller, *Shape replication through self-assembly and rnase enzymes*, Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, 2010, pp. 1045–1064.

2 Leonard Adleman, *Toward a mathematical theory of self-assembly (extended abstract)*, Tech. Report 00-722, University of Southern California, 2000.

3 Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang, *Running time and program size for self-assembled squares*, Proceedings of the thirty-third annual ACM Symposium on Theory of Computing, 2001, pp. 740–748.

4 Leonard Adleman, Qi Cheng, Ashish Goel, Ming-Deh Huang, and Hal Wasserman, *Linear self-assemblies: Equilibria, entropy and convergence rates*, In Sixth International Conference on Difference Equations and Applications, Taylor and Francis, 2001.

5 Robert D. Barish, Rebecca Schulman, Paul W. Rothemund, and Erik Winfree, *An information-bearing seed for nucleating algorithmic self-assembly*, Proceedings of the National Academy of Sciences **106** (2009), no. 15, 6054–6059.

6 Ho-Lin Chen and Ashish Goel, *Error free self-assembly with error prone tiles*, Proceedings of the 10th International Meeting on DNA Based Computers, 2004.

7 Ho-Lin Chen, Rebecca Schulman, Ashish Goel, and Erik Winfree, *Reducing facet nucleation during algorithmic self-assembly*, Nano Letters **7** (2007), no. 9, 2913–2919.

8 Qi Cheng, Gagan Aggarwal, Michael H. Goldwasser, Ming-Yang Kao, Robert T. Schweller, and Pablo Moisset de Espanés, *Complexities for generalized models of self-assembly*, SIAM Journal on Computing **34** (2005), 1493–1515.

9 Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Mashhood Ishaque, Eynat Rafalin, Robert T. Schweller, and Diane L. Souvaine, *Staged self-assembly: nanomanufacture of arbitrary shapes with $O(1)$ glues*, Natural Computing **7** (2008), no. 3, 347–370.

10 Erik D. Demaine, Matthew J. Patitz, Robert T. Schweller, and Scott M. Summers, *Self-assembly of arbitrary shapes using rnase enzymes: Meeting the kolmogorov bound with small scale factor (extended abstract)*, Tech. Report 1004.4383, Computing Research Repository, 2010.

11 David Doty, *Randomized self-assembly for exact shapes*, SIAM Journal on Computing **39** (2010), no. 8, 3521–3552.

12 David Doty, Matthew J. Patitz, Dustin Reishus, Robert T. Schweller, and Scott M. Summers, *Strong fault-tolerance for self-assembly with fuzzy temperature*, Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science, 2010, pp. 417–426.

13 James I. Lathrop, Jack H. Lutz, Matthew J. Patitz, and Scott M. Summers, *Computability and complexity in self-assembly*, Theory of Computing Systems, to appear.

14 Furong Liu, Ruojie Sha, and Nadrian C. Seeman, *Modifying the surface features of two-dimensional DNA crystals.*, Journal of the American Chemical Society **121** (1999), no. 5, 917–922.

15 Chris Luhrs, *Polyomino-safe DNA self-assembly via block replacement*, Proceedings of The Fourteenth International Meeting on DNA Computing and Molecular Programming (DNA 14) (Ashish Goel, Friedrich C. Simmel, and Petr Sosík, eds.), Lecture Notes in Computer Science, vol. 5347, Springer, 2008, pp. 112–126.

16 Chengde Mao, Thomas H. LaBean, John H. Relf, and Nadrian C. Seeman, *Logical computation using algorithmic self-assembly of DNA triple-crossover molecules.*, Nature **407** (2000), no. 6803, 493–6.

17 Chengde Mao, Weiqiong Sun, , and Nadrian C. Seeman, *Designed two-dimensional DNA holliday junction arrays visualized by atomic force microscopy.*, Journal of the American Chemical Society **121** (1999), no. 23, 5437–5443.

18 Paul W. K. Rothemund and Erik Winfree, *The program-size complexity of self-assembled squares (extended abstract)*, Proceedings of the thirty-second annual ACM Symposium on Theory of Computing, 2000, pp. 459–468.

19 Paul W.K. Rothemund, Nick Papadakis, and Erik Winfree, *Algorithmic self-assembly of DNA Sierpinski triangles*, PLoS Biology **2** (2004), no. 12, 2041–2053.

20 David Soloveichik and Erik Winfree, *Complexity of self-assembled shapes*, SIAM Journal on Computing **36** (2007), no. 6, 1544–1569.

21 Paul Vitányi and Ming Li, *An introduction to kolmogorov complexity and its applications*, Springer, 1997.

22 Erik Winfree, *Algorithmic self-assembly of DNA*, Ph.D. thesis, California Institute of Technology, June 1998.

23 ———, *Simulations of computing by self-assembly*, Tech. Report CaltechCSTR:1998.22, California Institute of Technology, 1998.

24 ———, *Self-healing tile sets*, Nanotechnology: Science and Computation (Junghuei Chen, Natasa Jonoska, and Grzegorz Rozenberg, eds.), Natural Computing Series, Springer, 2006, pp. 55–78.

25 Erik Winfree and Renat Bekbolatov, *Proofreading tile sets: Error correction for algorithmic self-assembly.*, DNA (Junghuei Chen and John H. Reif, eds.), Lecture Notes in Computer Science, vol. 2943, Springer, 2003, pp. 126–144.

26 Erik Winfree, Furong Liu, Lisa A. Wenzler, and Nadrian C. Seeman, *Design and self-assembly of two-dimensional DNA crystals.*, Nature **394** (1998), no. 6693, 539–44.

Weakly Unambiguous Morphisms

Dominik D. Freydenberger[1], Hossein Nevisi[2], and Daniel Reidenbach[3]

1 **Institut für Informatik, Goethe-Universität**
Frankfurt am Main, Germany
freydenberger@em.uni-frankfurt.de

2 **Department of Computer Science, Loughborough University**
Loughborough, Leicestershire LE11 3TU, United Kingdom
H.Nevisi@lboro.ac.uk
Corresponding author

3 **Department of Computer Science, Loughborough University**
Loughborough, Leicestershire LE11 3TU, United Kingdom
D.Reidenbach@lboro.ac.uk

Abstract

A nonerasing morphism σ is said to be weakly unambiguous with respect to a word w if σ is the only nonerasing morphism that can map w to $\sigma(w)$, i. e., there does not exist any other nonerasing morphism τ satisfying $\tau(w) = \sigma(w)$. In the present paper, we wish to characterise those words with respect to which there exists such a morphism. This question is nontrivial if we consider so-called length-increasing morphisms, which map a word to an image that is strictly longer than the word. Our main result is a compact characterisation that holds for all morphisms with ternary or larger target alphabets. We also comprehensively describe those words that have a weakly unambiguous length-increasing morphism with a unary target alphabet, but we have to leave the problem open for binary alphabets, where we can merely give some non-characteristic conditions.

1998 ACM Subject Classification G.2.m [Discrete Mathematics]: Miscellaneous

Keywords and phrases Combinatorics on words, Morphisms, Ambiguity

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.213

1 Introduction

For any alphabets $\mathcal{A}$ and $\mathcal{B}$, a morphism $\sigma : \mathcal{A}^* \to \mathcal{B}^*$ is said to be *ambiguous* with respect to a word s if there exists a second morphism $\tau : \mathcal{A}^* \to \mathcal{B}^*$ mapping s to the same image as σ; if such a morphism τ does not exist, then σ is called *unambiguous* (with respect to s). For example, if we consider $\mathcal{A} := \{A, B, C\}$, $\mathcal{B} := \{a, b\}$ and $s := A\,B\,B\,C\,A\,C$, then the morphism σ, defined by $\sigma(A) := abb$, $\sigma(B) := abbb$, $\sigma(C) := abbbb$, is ambiguous with respect to s, since there exists a different morphism τ, given by $\tau(A) := abbab$, $\tau(B) := bbab$, $\tau(C) := bbb$, satisfying $\tau(s) = \sigma(s)$:

$$\begin{gathered}\overbrace{a\;b\;b}^{\sigma(A)}\;\overbrace{a\;b\;b\;b}^{\sigma(B)}\;\overbrace{a\;b\;b\;b}^{\sigma(B)}\;\overbrace{a\;b\;b\;b\;b}^{\sigma(C)}\;\overbrace{a\;b\;b}^{\sigma(A)}\;\overbrace{a\;b\;b\;b\;b}^{\sigma(C)}\\ \underbrace{a\;b\;b\;a\;b}_{\tau(A)}\;\underbrace{b\;b\;a\;b}_{\tau(B)}\;\underbrace{b\;b\;a\;b}_{\tau(B)}\;\underbrace{b\;b\;b}_{\tau(C)}\;\underbrace{a\;b\;b\;a\;b}_{\tau(A)}\;\underbrace{b\;b\;b}_{\tau(C)}\,.\end{gathered}$$

In contrast to this, as can be verified with little effort, the morphism $\sigma' : \mathcal{A}^* \to \mathcal{B}^*$, defined by $\sigma'(A) := \sigma'(C) := a$ and $\sigma'(B) := b$, is unambiguous with respect to s.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 213–224
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

The potential ambiguity of morphisms is not only a fundamental phenomenon in combinatorics on words, but it also shows connections to various concepts in computer science. This particularly holds for *equality sets* (and, hence, the *Post Correspondence Problem*, see Harju and Karhumäki [5]) and *pattern languages* (see Mateescu and Salomaa [7]). Regarding the latter topic, insights into the ambiguity of morphisms have been used to solve a number of prominent problems (see, e. g., Reidenbach [8, 9, 10]), revealing that unambiguous morphisms, in a setting where various morphisms are applied to the same word, have the ability to optimally encode information about the structure of the word. This shows an interesting contrast to the foundations of coding theory (see Berstel and Perrin [1]), which is based on *injective* morphisms.

Since unambiguity can, thus, be seen as a desirable property of morphisms, the initial work on this topic by Freydenberger, Reidenbach and Schneider [3] and most of the subsequent papers have focused on the following question:

▶ **Problem 1.** Let s be a word over an arbitrary alphabet. Does there *exist* a morphism (preferably with a finite target alphabet comprising at least two letters) that is *unambiguous* with respect to s?

In order to further qualify this problem, [3] introduces two types of unambiguity: The first type follows our intuitive definition given above; more precisely, a morphism σ is called *strongly unambiguous* with respect to a word s if it there exists no morphism τ satisfying $\tau(s) = \sigma(s)$ and, for a symbol x occurring in s, $\tau(x) \neq \sigma(x)$. The second type slightly relaxes this requirement by calling σ *weakly unambiguous* with respect to s if there is no *nonerasing* morphism τ (which means that τ must not map any symbol to the empty word) showing the above properties. Thus, e. g., our initial example morphism σ is weakly unambiguous with respect to $s' := A\,A\,B$, but it is not strongly unambiguous, since the morphism τ, given by $\tau(A) := \varepsilon$ and $\tau(B) := \sigma(s')$ (where ε stands for the empty word), satisfies $\tau(s') = \sigma(s')$. By definition, every strongly unambiguous *nonerasing* morphism is also weakly unambiguous, but – as shown by this example – the converse does not necessarily hold.

Apart from some very basic considerations, previous research has focussed on *strongly* unambiguous morphisms, partly giving comprehensive results on their existence; positive results along this line then automatically also hold for weak unambiguity. Freydenberger et al. [3] characterise those words with respect to which there exist strongly unambiguous nonerasing morphisms, and their characteristic criterion reveals that the existence of such morphisms is equivalent to a number of other vital properties of words, such as being a fixed point of a nontrivial morphism (see, e. g., Hamm and Shallit [4]) or being a shortest generator of a terminal-free E-pattern language. Freydenberger and Reidenbach [2], among other results, improve and deepen the techniques used in [3]. Schneider [12] studies the more general problem of the existence of arbitrary (i. e., possibly erasing) strongly unambiguous morphisms. While [12] provides a characterisation of those words that have a strongly unambiguous erasing morphism with an infinite target alphabet, a comprehensive result on finite target alphabets is still open. It is known, however, that a distinct characteristic criterion is required for every alphabet size (unlike the restricted problem for strongly unambiguous nonerasing morphisms, the existence of which can be characterised for all non-unary alphabets identically), and that each of these criteria is NP-hard. Reidenbach and Schneider [11] continue this strand of research, demonstrating that the existence of strongly unambiguous erasing morphisms is closely related to decision problems for multi-pattern languages, and they show that the same criterion that characterises the existence of such morphisms for infinite target alphabets also, for all binary or larger alphabets, characterises the existence of erasing morphisms with a strongly restricted ambiguity.

In the present paper, we wish to investigate the existence of *weakly* unambiguous nonerasing morphisms; in other words, we initiate the research on the ambiguity of morphisms in free semigroups without empty word. When considering this problem as indicated above, we can already refer to a strong yet trivial insight mentioned by Freydenberger et al. [3], stating that there indeed is a weakly unambiguous morphism with respect to every word. More precisely, it directly follows from the definitions that every 1-uniform morphism (i. e., a morphism that maps each variable in the pattern to a word of length 1) is weakly unambiguous with respect to every word. Despite this immediate and unexciting observation, weak unambiguity deserves further research, since there are major fields of study that are exclusively based on nonerasing morphisms; this particularly holds for pattern languages, where so-called *nonerasing* (or *NE* for short) pattern languages have been intensively investigated. We therefore exclude the 1-uniform morphisms from our considerations and study *length-increasing* nonerasing morphisms instead, i. e., we deal with morphisms σ that, for the word s they are applied to, satisfy $|\sigma(s)| > |s|$. Hence, we wish to examine the following problem:

▶ **Problem 2.** Let s be a word over an arbitrary alphabet. Does there exist a *length-increasing* nonerasing morphism that is *weakly* unambiguous with respect to s?

Our results in the present paper shall provide a nearly comprehensive answer to this question, demonstrating that a combinatorially rich theory results from it. In particular, we show that the existence of weakly unambiguous length-increasing morphisms depends on the size of the target alphabet considered. However, unlike the above-mentioned result by Schneider [12] on the existence of strongly unambiguous erasing morphisms, we can give a compact and efficiently decidable characteristic condition on Problem 2, which holds for *all* target alphabets that consist of at least three letters and which describes a type of words we believe has not been discussed in the literature so far. Interestingly, this characterisation does not hold for binary target alphabets. In this case, we can give a number of strong conditions, but still do not even know whether Problem 2 is decidable. In contrast to this phenomenon, it is of course not surprising that for unary target alphabets again a different approach is required. Regarding this specification of Problem 2, we shall give a characteristic condition.

Note that, due to space constraints, almost all proofs and some related definitions and lemmas are omitted from this paper.

2 Definitions

Let $\mathbb{N} := \{1, 2, 3, ...\}$ and Σ be alphabets. We call any symbol in $\mathbb{N}$ a variable and any symbol in Σ a letter. For the *concatenation* of two words w_1, w_2, we write $w_1 \cdot w_2$ or simply $w_1 w_2$. The word that results from $n-$fold concatenation of a word w is denoted by w^n. The notion $|x|$ stands for the size of a set x or the length of a word x. We denote the empty word by ε, i. e., $|\varepsilon| = 0$. The symbol [...] is used to omit some canonically defined parts of a given word, e. g., $\alpha = 1 \cdot 2 \cdot [...] \cdot 5$ stands for $\alpha = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5$. In order to distinguish between a word over $\mathbb{N}$ and a word over Σ, we call the former a *pattern*. We name patterns with lower case letters from the beginning of the Greek alphabet such as α, β, γ. For every alphabet A, A^* is the set of all (empty and non-empty) words over A, and $A^+ := A^* \setminus \{\varepsilon\}$. We call a word $v \in A^*$ a *factor* of a word $w \in A^*$ if, for some $u_1, u_2 \in A^*$, $w = u_1 v u_2$; moreover, if v is a factor of w then we say that w contains v and denote this by $v \sqsubseteq w$. If $v \neq w$, then we say that v is a *proper* factor of w and denote this by $v \sqsubset w$. If $u_1 = \varepsilon$, then v is a prefix of w, and if $u_2 = \varepsilon$, then v is a suffix of w.

With regard to an arbitrary pattern α, $\mathrm{var}(\alpha)$ denotes the set of all variables occurring in α, and $|\alpha|_\beta$, $\beta \sqsubseteq \alpha$, shows the number of (possibly overlapping) occurrences of β in α.

A *morphism* is a mapping that is compatible with concatenation, i. e., for patterns $\alpha \in \mathbb{N}^+$ and $\beta \in \mathbb{N}^+$, a morphism $\sigma : \mathbb{N}^+ \to \Sigma^*$ satisfies $\sigma(\alpha \cdot \beta) = \sigma(\alpha) \cdot \sigma(\beta)$. A morphism σ is called *nonerasing* provided that, for every $i \in \mathbb{N}$, $\sigma(i) \neq \varepsilon$. The morphism σ is *length-increasing* if $|\sigma(\alpha)| > |\alpha|$, and it is called *1-uniform* if, for every $i \in \mathbb{N}$, $|\sigma(i)| = 1$.

For any alphabet Σ, for any morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$ and for any pattern $\alpha \in \mathbb{N}^+$, we call σ *weakly unambiguous* with respect to α if there is no morphism $\tau : \mathbb{N}^+ \to \Sigma^+$ with $\tau(\alpha) = \sigma(\alpha)$ and, for some $q \in \mathrm{var}(\alpha)$, $\tau(q) \neq \sigma(q)$. Moreover, for any morphism $\sigma : \mathbb{N}^* \to \Sigma^*$, σ is said to be *strongly unambiguous* with respect to α, if there is no morphism $\tau : \mathbb{N}^* \to \Sigma^*$ with $\tau(\alpha) = \sigma(\alpha)$ and, for some $q \in \mathrm{var}(\alpha)$, $\tau(q) \neq \sigma(q)$. On the other hand, σ is *ambiguous* with respect to α, if there is a morphism $\tau : \mathbb{N}^+ \to \Sigma^+$ with $\tau(\alpha) = \sigma(\alpha)$ and, for some $q \in \mathrm{var}(\alpha)$, $\tau(q) \neq \sigma(q)$.

We call any pattern $\alpha \in \mathbb{N}^+$ *prolix* if and only if, there exists a decomposition $\alpha = \beta_0\gamma_1\beta_1\gamma_2\beta_2[...]\beta_{n-1}\gamma_n\beta_n$ with $n \geq 1$, $\beta_k \in \mathbb{N}^*$ and $\gamma_k \in \mathbb{N}^*$, $k \leq n$, such that

1. for every k, $1 \leq k \leq n$, $|\gamma_k| \geq 2$,
2. for every k, $1 \leq k \leq n$ and, for every k', $0 \leq k' \leq n$, $\mathrm{var}(\gamma_k) \cap \mathrm{var}(\beta_{k'}) = \emptyset$,
3. for every k, $1 \leq k \leq n$, there exists an $i_k \in \mathrm{var}(\gamma_k)$ such that $|\gamma_k|_{i_k} = 1$ and, for every k', $1 \leq k' \leq n$, if $i_k \in \mathrm{var}(\gamma_{k'})$ then $\gamma_k = \gamma_{k'}$.

We call $\alpha \in \mathbb{N}^+$ *succinct* if and only if it is not prolix. Thus, for example, the pattern $1 \cdot 2 \cdot 3 \cdot 2 \cdot 4 \cdot 2 \cdot 1 \cdot 5 \cdot 5 \cdot 4 \cdot 2 \cdot 1 \cdot 1 \cdot 2 \cdot 3 \cdot 2$ is prolix (with $\beta_0 := \varepsilon$, $\gamma_1 := 1 \cdot 2 \cdot 3 \cdot 2$, $\beta_1 := \varepsilon$, $\gamma_2 := 4 \cdot 2 \cdot 1$, $\beta_2 := 5 \cdot 5$, $\gamma_3 := 4 \cdot 2 \cdot 1$, $\beta_3 := \varepsilon$, $\gamma_4 := 1 \cdot 2 \cdot 3 \cdot 2$, $\beta_4 := \varepsilon$), whereas $1 \cdot 2 \cdot 3 \cdot 3 \cdot 4 \cdot 2 \cdot 4 \cdot 2 \cdot 1$ is succinct.

3 Loyal neighbours

Before we begin our examination of Problem 2, we introduce some notions on structural properties of variables in patterns that shall be used in the subsequent sections.

In our first definition, we introduce a concept that collects the neighbours of a variable in a pattern.

▶ **Definition 3.** Let $\alpha \in \mathbb{N}^+$. For every $j \in \mathrm{var}(\alpha)$, we define the following sets:

$$L_j := \{k \in \mathrm{var}(\alpha) \mid \alpha = ... \cdot k \cdot j \cdot ...\},$$
$$R_j := \{k \in \mathrm{var}(\alpha) \mid \alpha = ... \cdot j \cdot k \cdot ...\}.$$

Also, if $\alpha = j...$, then $\varepsilon \in L_j$, and if $\alpha = ...j$, then $\varepsilon \in R_j$.

Thus, the notation L_j refers to all left neighbours of variable j and R_j to all right neighbours of j. To illustrate these notions, we give an example.

▶ **Example 4.** We consider $\alpha := 1 \cdot 2 \cdot 3 \cdot 1 \cdot 4 \cdot 5 \cdot 6 \cdot 1 \cdot 4 \cdot 7 \cdot 8$. For the variable 1, we have $L_1 = \{\varepsilon, 3, 6\}$ and $R_1 = \{2, 4\}$.

We now introduce the concept of loyalty of neighbouring variables, which is vital for the examination of weakly unambiguous morphisms.

▶ **Definition 5.** Let $\alpha \in \mathbb{N}^+$. A variable $i \in \mathrm{var}(\alpha)$ has *loyal neighbours (in α)* if and only if at least one of the following cases is satisfied:

1. $\varepsilon \notin L_i$ and, for every $j \in L_i$, $R_j = \{i\}$, or
2. $\varepsilon \notin R_i$ and, for every $j \in R_i$, $L_j = \{i\}$.

Using the above definition, we can divide the variables of any pattern into two sets.

▶ **Definition 6.** For any pattern $\alpha \in \mathbb{N}^+$, $|\alpha| > 1$, let S_α be the set of variables that have loyal neighbours and E_α be the set of variables that do not have loyal neighbours in α.

The following example clarifies the mentioned definitions.

▶ **Example 7.** Let $\alpha := 1 \cdot 2 \cdot \mathbf{3} \cdot \mathbf{4} \cdot 5 \cdot 6 \cdot \mathbf{4} \cdot \mathbf{3} \cdot 7 \cdot 8$. Definition 3 implies that

$$\begin{aligned} &L_1 = \{\varepsilon\},\ L_2 = \{1\},\ L_3 = \{2,4\},\ L_4 = \{3,6\},\\ &L_5 = \{4\},\ L_6 = \{5\},\ L_7 = \{3\},\quad L_8 = \{7\},\\ &R_1 = \{2\},\ R_2 = \{3\},\ R_3 = \{4,7\},\ R_4 = \{5,3\},\\ &R_5 = \{6\},\ R_6 = \{7\},\ R_7 = \{8\},\quad R_8 = \{\varepsilon\}. \end{aligned}$$

According to Definition 5, the variables 3 and 4 do not have loyal neighbours. Thus, due to Definition 6, $S_\alpha = \{1,2,5,6,7,8\}$ and $E_\alpha = \{3,4\}$.

Freydenberger et al. [3] demonstrate that the partition of the set of all patterns into succinct and prolix ones is characteristic for the existence of strongly unambiguous nonerasing morphisms:

▶ **Theorem 8** (Freydenberger et al. [3]). *Let $\alpha \in \mathbb{N}^*$, let Σ be an alphabet, $|\Sigma| \geq 2$. There exists a strongly unambiguous nonerasing morphism $\sigma : \mathbb{N}^* \to \Sigma^*$ with respect to α if and only if α is succinct.*

Our subsequent remark shows that having a variable with loyal neighbours is a sufficient, but not a necessary condition for a pattern being prolix.

▶ **Proposition 9.** *Let $\alpha \in \mathbb{N}^+$. If $S_\alpha \neq \emptyset$, then α is prolix. In general, the converse of this statement does not hold true.*

4 Weakly unambiguous morphisms with $|\Sigma| \geq 3$

We now make use of the concepts introduced in the previous section to comprehensively solve Problem 2 for all but unary and binary target alphabets of the morphisms.

We start this section by giving some lemmas that are required when proving the main results of this paper. The first lemma is a general combinatorial insight that can be used in the proof of Lemma 11 – which, in turn, is a fundamental lemma in this paper.

▶ **Lemma 10.** *Let v be a word and n a natural number. If, for a word w, w^n is a proper factor of v^n, then w is a proper factor of v.*

We continue our studies with the following lemma, which is a vital tool for the proof of many statements of this paper. It features an important property of two different morphisms that map a pattern to the same image.

▶ **Lemma 11.** *Let $\alpha \in \mathbb{N}^+$, $|\alpha| > 1$. Assume that $\sigma : \mathbb{N}^+ \to \Sigma^+$ is a morphism such that, for an $i \in \operatorname{var}(\alpha)$, $|\sigma(i)| > 1$ and, for every $x \in \operatorname{var}(\alpha) \setminus \{i\}$, $|\sigma(x)| = 1$. Moreover, assume that τ is a nonerasing morphism satisfying $\tau(\alpha) = \sigma(\alpha)$. If there exists a $j \in \operatorname{var}(\alpha)$ with $\tau(j) \neq \sigma(j)$, then $\tau(i) \sqsubset \sigma(i)$.*

The next lemma, which directly results from Definition 5, discusses those patterns having at least one square; more precisely, there exists an $i \in \mathbb{N}$ with $i^2 \sqsubset \alpha$.

▶ **Lemma 12.** *Let $\alpha \in \mathbb{N}^+$. If, for an $i \in \mathbb{N}$, $i^2 \sqsubseteq \alpha$, then $i \in E_\alpha$.*

The subsequent characterisation of those patterns that have a weakly unambiguous length-increasing morphism with ternary or larger target alphabets is the main result of this paper. It yields a novel partition of the set of all patterns over any sub-alphabet of $\mathbb{N}$. This partition is different from the partition into prolix and succinct patterns, which characterises the existence of strongly unambiguous nonerasing morphisms (see Theorem 8).

▶ **Theorem 13.** *Let $\alpha \in \mathbb{N}^+$ with $|\alpha| > 1$ and let $|\Sigma| \geq 3$. There is a weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$ with respect to α if and only if E_α is not empty.*

Proof. Let $\{a, b, c\} \subseteq \Sigma$.
We begin with the *if* direction. Assume that E_α is not empty. This means that there is at least one variable $i \in \mathrm{var}(\alpha)$ that does not have loyal neighbours, i. e., $i \in E_\alpha$. Due to Definition 5 and Lemma 12, one of the following cases is satisfied:

Case 1: $i^2 \sqsubseteq \alpha$.
We define a morphism σ by $\sigma(x) := bc$ if $x = i$ and $\sigma(x) := a$ if $x \neq i$. So, $\sigma(i^2) = bcbc$. According to Lemma 11, any nonerasing morphism $\tau : \mathbb{N}^+ \to \Sigma^+$ with $\tau(\alpha) = \sigma(\alpha)$ and, for some $k \in \mathrm{var}(\alpha)$, $\tau(k) \neq \sigma(k)$, must satisfy $\tau(i) \neq \sigma(i)$, and this means that $\tau(i)$ should be a proper factor of $\sigma(i)$. This implies that $\tau(i) = b$ or $\tau(i) = c$ and as a result, $\tau(i^2) = bb$ or $\tau(i^2) = cc$. Since $\sigma(\alpha)$ does not contain the factors bb and cc, we can conclude that $\tau(\alpha) \neq \sigma(\alpha)$ and consequently, σ is weakly unambiguous with respect to α.

Case 2: $i^2 \not\sqsubseteq \alpha$, and one of the following cases is satisfied:
Case 2.1: If $\varepsilon \notin L_i$, then there exists a $j \in L_i$ such that $R_j \neq \{i\}$, and if $\varepsilon \notin R_i$, then there exits a $j' \in R_i$ such that $L_{j'} \neq \{i\}$.
Case 2.2: $\varepsilon \in L_i$ and $\varepsilon \in R_i$.

Let $\sigma : \mathbb{N}^+ \to \{a, b, c\}^+$ be the morphism defined in Case 1. Due to Lemma 11, any morphism $\tau : \mathbb{N}^+ \to \Sigma^+$ with $\tau(\alpha) = \sigma(\alpha)$ and, for some $k \in \mathrm{var}(\alpha)$, $\tau(k) \neq \sigma(k)$, must satisfy $\tau(i) \neq \sigma(i)$, and this means that $\tau(i)$ should be a proper factor of $\sigma(i)$. Thus, $\tau(i) = b$ or $\tau(i) = c$.
With regard to Case 2.1, consider $\tau(i) = c$, and $\varepsilon \notin L_i$. Due to the number of c in $\sigma(\alpha)$, which equals the number of occurrences of i in α, and also due to $\sigma(i) = bc$, the positions of c of $\tau(i)$ should be at the same positions as c of $\sigma(i)$ in $\sigma(\alpha)$. So, to have $\tau(\alpha) = \sigma(\alpha)$, for every $l \in L_i$, b is a suffix of $\tau(l)$, and as a result b is suffix of $\tau(j)$. However, since $R_j \neq \{i\}$, the number of occurrences of b in $\tau(\alpha)$ is greater than the number of occurrences of b in $\sigma(\alpha)$. Hence, $\tau(\alpha) \neq \sigma(\alpha)$. Consider $\tau(i) = b$, and $\varepsilon \notin R_i$. Due to the number of b in $\sigma(\alpha)$, which equals the number of occurrences of i in α, and also due to $\sigma(i) = bc$, the positions of b of $\tau(i)$ should be at the same positions as b of $\sigma(i)$ in $\sigma(\alpha)$. Hence, to have $\tau(\alpha) = \sigma(\alpha)$, for every $r \in R_i$, c is a prefix of $\tau(r)$, and consequently, c is prefix of $\tau(j')$. However, since $L_{j'} \neq \{i\}$, the number of occurrences of c in $\tau(\alpha)$ is greater than the number of occurrences of c in $\sigma(\alpha)$. This again implies $\tau(\alpha) \neq \sigma(\alpha)$.
Case 2.2 means that $\alpha = i\alpha' i$, $\alpha' \in \mathbb{N}^*$. So, $\sigma(\alpha) = bc\sigma(\alpha')bc$. As mentioned above, due to Lemma 11, $\tau(i) = b$ or $\tau(i) = c$. This implies that $\tau(\alpha)$ starts with b and finishes with b, or it starts with c and finishes with c. Thus, $\tau(\alpha) \neq \sigma(\alpha)$. Hence, we can conclude that if $E_\alpha \neq \emptyset$, then there is a weakly unambiguous length-increasing morphism with respect to α.

We now prove the *only if* direction. In fact, we want to show that if there is a weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$ with respect to α, then E_α is not empty. Assume that σ maps one of the variables of α to a word of length more than 1, and let this variable be i. Also, let $\sigma(i) := a_1a_2[...]a_n$ with $n \geq 2$ and, for every q, $1 \leq q \leq n$, $a_q \in \Sigma$. Assume to the contrary that E_α is empty. Thus, due to Lemma 12, $i^2 \not\sqsubseteq \alpha$. According to Definition 5, one of the following cases is satisfied:

Case 1: $\varepsilon \notin L_i$ and, for every $j \in L_i$, $R_j = \{i\}$.
From this condition, we can directly conclude that

$$\alpha := \alpha_1 \cdot l_1 \cdot i \cdot \alpha_2 \cdot l_2 \cdot i \cdot [...] \cdot \alpha_m \cdot l_m \cdot i \cdot \alpha_{m+1},$$

with $|\alpha|_i = m$ and, for every k, $1 \leq k \leq m$ and, for every k', $1 \leq k' \leq m+1$, $l_k \in L_i$, $\alpha_{k'} \in \mathbb{N}^*$, $i \neq l_k$ and, $i, l_k \notin \mathrm{var}(\alpha_{k'})$. Thus,

$$\begin{aligned}\sigma(\alpha) &= \sigma(\alpha_1)\sigma(l_1)\, a_1a_2[...]a_n \cdot \sigma(\alpha_2)\sigma(l_2)a_1a_2[...]a_n \\ &\quad \cdot[...] \cdot \sigma(\alpha_m)\sigma(l_m)\, a_1a_2[...]a_n \cdot \sigma(\alpha_{m+1})\end{aligned}$$

We now define the nonerasing morphism τ such that, for every k, $1 \leq k \leq m$, $\tau(l_k) := \sigma(l_k)a_1$, $\tau(i) := a_2a_3[...]a_n$ and, for all other variables in α, τ is identical to σ. Due to the fact that, for every k, $1 \leq k \leq m$, $R_{l_k} = \{i\}$, we can conclude that $\tau(\alpha) = \sigma(\alpha)$; since τ is nonerasing, σ is not weakly unambiguous.

Case 2: $\varepsilon \notin R_i$ and, for every $j \in R_i$, $L_j = \{i\}$.
We can directly conclude that $\alpha := \alpha_1 \cdot i \cdot r_1 \cdot \alpha_2 \cdot i \cdot r_2 \cdot [...] \cdot \alpha_m \cdot i \cdot r_m \cdot \alpha_{m+1}$, with $|\alpha|_i = m$ and, for every k, $1 \leq k \leq m$ and, for every k', $1 \leq k' \leq m+1$, $r_k \in R_i$, $\alpha_{k'} \in \mathbb{N}^*$, $i \neq r_k$, and $i, r_k \notin \mathrm{var}(\alpha_{k'})$. So,

$$\begin{aligned}\sigma(\alpha) &= \sigma(\alpha_1)a_1a_2[...]a_n\sigma(r_1) \cdot \sigma(\alpha_2)a_1a_2[...]a_n\sigma(r_2) \\ &\quad \cdot[...] \cdot \sigma(\alpha_m)a_1a_2[...]a_n\sigma(r_m\cdot)\sigma(\alpha_{m+1})\end{aligned}$$

We now define the nonerasing morphism τ such that, for every k, $1 \leq k \leq m$, $\tau(r_k) := a_n\sigma(r_k)$ and $\tau(i) := a_1a_2[...]a_{n-1}$ and, for all other variables in α, τ is identical to σ. As, for every k, $1 \leq k \leq m$, $L_{r_k} = \{i\}$, we can conclude that $\tau(\alpha) = \sigma(\alpha)$; since τ is nonerasing, σ is not weakly unambiguous. Hence, $E_\alpha = \emptyset$ implies that σ is not weakly unambiguous, which contradicts the assumption. Consequently, E_α is not empty. ◀

In order to illustrate Theorem 13, we give two examples:

▶ **Example 14.** Let $\alpha := 1 \cdot 2 \cdot 3 \cdot 4 \cdot 1 \cdot 2 \cdot 3$. According to Definition 6, $S_\alpha = \{1, 2, 3\}$ and $E_\alpha = \{4\}$. In other words, the variable 4 does not have loyal neighbours. We define a morphism σ by $\sigma(4) := bc$ and, for every other variable $j \in \mathrm{var}(\alpha)$, $\sigma(j) := a$. Due to Lemma 11, any morphism τ with $\tau(\alpha) = \sigma(\alpha)$ and, for a $k \in \mathrm{var}(\alpha)$, $\tau(k) \neq \sigma(k)$ needs to split the factor bc. Hence, $\tau(1)$ needs to contain c, or $\tau(3)$ needs to contain b. However, since $|\alpha|_1 = 2$ and $|\alpha|_3 = 2$, $|\tau(\alpha)|_c > |\sigma(\alpha)|_c$, or $|\tau(\alpha)|_b > |\sigma(\alpha)|_b$. Consequently, $\tau(\alpha) \neq \sigma(\alpha)$ and as a result, σ is weakly unambiguous with respect to α.

▶ **Example 15.** Let $\alpha := 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 4 \cdot 7 \cdot 8 \cdot 3$. According to Definition 5, all variables have loyal neighbours, or in other words, $E_\alpha = \emptyset$. Hence, it follows from Theorem 13 that there is no weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \rightarrow \Sigma^+$, $|\Sigma| \geq 3$, with respect to α.

We now give an alternative version of Theorem 13 that is based on regular expressions.

▶ **Corollary 16.** *Let $\alpha \in \mathbb{N}^+$ and let Σ be an alphabet, $|\Sigma| \geq 3$. There is no weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \rightarrow \Sigma^+$ with respect to α if and only if, for every $i \in \mathrm{var}(\alpha)$, at least one of the following statements is satisfied:*

- *there exists a partition* $L, N, \{i\}$ *of* $\mathrm{var}(\alpha)$ *such that* $\alpha \in (N^* L i)^+ N^*$*,*
- *there exists a partition* $R, N, \{i\}$ *of* $\mathrm{var}(\alpha)$ *such that* $\alpha \in (N^* i R)^+ N^*$*.*

We conclude this section by determining the complexity of the decision problem resulting from Theorem 13.

▶ **Theorem 17.** *Let* $\alpha \in \mathbb{N}^+$ *with* $|\alpha| > 1$*, and let* $|\Sigma| \geq 3$*. The problem of whether there is a weakly unambiguous length-increasing morphism* $\sigma : \mathbb{N}^+ \to \Sigma^+$ *with respect to* α *is decidable in polynomial time.*

Hence, the complexity of Problem 2 is comparable to that of the equivalent problem for strongly unambiguous nonerasing morphisms (this is a consequence of the characterisation by Freydenberger et al. [3] and the complexity consideration by Holub [6]). In contrast to this, when we ask for the existence of strongly unambiguous erasing morphisms, the problem is NP-hard (according to Schneider [12]).

5 Weakly unambiguous morphisms with $|\Sigma| = 2$

As we shall demonstrate below, our characterisation in Theorem 13 does not hold for binary target alphabets Σ (see Corollary 24). Hence, we have to study this case separately. The most significant result of our considerations is a necessary condition on the structure of those patterns α that satisfy $E_\alpha \neq \emptyset$, but nevertheless do not have a weakly unambiguous morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$, $|\Sigma| = 2$.

Despite being restricted to ternary or larger alphabets, Theorem 13 and its proof have two important implications that also hold for unary and binary alphabets. The first of them shows that E_α being empty for any given pattern α is a sufficient condition for α not having any weakly unambiguous length-increasing morphism:

▶ **Corollary 18.** *Let* $\alpha \in \mathbb{N}^+$*, and let* Σ *be any alphabet. If* $E_\alpha = \emptyset$*, then there is no weakly unambiguous length-increasing morphism* $\sigma : \mathbb{N}^+ \to \Sigma^+$ *with respect to* α*. In general, the converse of this statement does not hold true.*

Hence, if we wish to characterise those patterns with respect to which there is a weakly unambiguous morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$, $|\Sigma| \leq 2$, then we can safely restrict our considerations to those patterns α where E_α is a nonempty set.

The second implication of Theorem 13 demonstrates that any length-increasing morphism that is weakly unambiguous with respect to a pattern α must have a particular, and very simple, shape for all variables in S_α:

▶ **Corollary 19.** *Let* $\alpha \in \mathbb{N}^+$*, let* Σ *be any alphabet, and let* $\sigma : \mathbb{N}^+ \to \Sigma^+$ *be a length-increasing morphism that is weakly unambiguous with respect to* α*. Then, for every* $i \in S_\alpha$*,* $|\sigma(i)| = 1$*.*

Thus, any weakly unambiguous length-increasing morphism with respect to a pattern α must not be length-increasing for the variables in S_α. This insight is very useful when searching for morphisms that might be weakly unambiguous with respect to a given pattern.

As shown by Corollary 18, if E_α is empty, then there is no weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$ with respect to α. In the next step, we give a strong *necessary* condition on the structure of those patterns α that satisfy $E_\alpha \neq \emptyset$, but nevertheless do *not* have a weakly unambiguous morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$, $|\Sigma| = 2$.

▶ **Theorem 20.** *Let $\alpha \in \mathbb{N}^+$ such that E_α is nonempty. Let Σ be an alphabet, $|\Sigma| = 2$. If there is no weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$ with respect to α, then for every $e \in E_\alpha$ there exists an $e' \in E_\alpha$, $e' \neq e$, such that $e \cdot e'$ and $e' \cdot e$ are factors of α.*

Theorem 20 (when compared to Theorem 13) provides deep insights into the difference between binary and ternary target alphabets if the weak unambiguity of morphisms is studied. In addition to this, it implies that whenever, for a given pattern $\alpha \in \mathbb{N}^+$ with $E_\alpha \neq \emptyset$, there exists an $e \in E_\alpha$ such that, for every $e' \in E_\alpha$ with $e' \neq e$, the factors $e \cdot e'$ or $e' \cdot e$ do not occur in α, then there is a weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$, $\Sigma = \{a, b\}$, with respect to α. It must be noted, though, that Theorem 20 does not describe a sufficient condition for the non-existence of weakly unambiguous length-increasing morphisms in case of $|\Sigma| = 2$; this is easily demonstrated by the pattern $1 \cdot 2 \cdot 1$ and further illustrated by Example 26.

As can be concluded from Example 7 and Theorem 13, there is a weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$, $|\Sigma| \geq 3$, with respect to $\alpha = 1\cdot 2\cdot \mathbf{3} \cdot \mathbf{4}\cdot 5\cdot 6\cdot \mathbf{4} \cdot \mathbf{3}\cdot 7\cdot 8$. We can define σ by $\sigma(3) := bc$ and, for every $j \neq 3$, $\sigma(j) := a$. In contrast to this, the next theorem implies that there is no weakly unambiguous morphism with respect to α if $|\Sigma| = 2$. In order to substantiate this theorem, we need the following lemma.

▶ **Lemma 21.** *Let Σ be an alphabet with $|\Sigma| = 2$, and let $\sigma : \mathbb{N}^+ \to \Sigma^+$ be a morphism. For all $x_1, x_2 \in \mathbb{N}$, there exist $a_1 \sqsubseteq \sigma(x_1)$ and $a_2 \sqsubseteq \sigma(x_2)$ such that $a_1 a_2 \sqsubseteq \sigma(x_1 \cdot x_2)$ and $a_2 a_1 \sqsubseteq \sigma(x_2 \cdot x_1)$.*

The next result introduces a *sufficient* condition on the *non-existence* of weakly unambiguous length-increasing morphisms $\sigma : \mathbb{N}^+ \to \Sigma^+$, $|\Sigma| = 2$. According to Theorem 20, it is necessary for the non-existence of such morphisms, with respect to a given pattern $\alpha \in \mathbb{N}^+$ that, for every $e \in E_\alpha$, there exists an $e' \in E_\alpha$, $e' \neq e$, such that $e \cdot e'$ and $e' \cdot e$ are factors of α. Hence, this requirement must be satisfied in the following theorem.

▶ **Theorem 22.** *Let $\alpha \in \mathbb{N}^+$ satisfying $E_\alpha \neq \emptyset$. Let Σ be an alphabet with $|\Sigma| = 2$. There is no weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$ with respect to α if*

1. *for every $e \in E_\alpha$, $e^2 \not\sqsubseteq \alpha$, and there is exactly one $e' \in E_\alpha \setminus \{e\}$ such that $e' \in L_e$ or $e' \in R_e$, $e' \cdot e \cdot e' \not\sqsubseteq \alpha$, and there are $s_1, s_2, s_3, s_4 \in S_\alpha$ such that $s_1 \cdot e \cdot e' \cdot s_2$ and $s_3 \cdot e' \cdot e \cdot s_4$ are factors of α,*
2. *for every $e \in E_\alpha$, $\varepsilon \notin R_e$ and $\varepsilon \notin L_e$,*
3. *for any $s, s' \in S_\alpha$ and $e, e' \in E_\alpha$, if $(s \cdot e \cdot e' \cdot s') \sqsubset \alpha$, then, for all occurrences of s and s' in α, the right neighbour of s is the factor $e \cdot e'$ and the left neighbour of s' is the factor $e \cdot e'$, and*
4. *for any $s, s' \in S_\alpha$ and $e \in E_\alpha$, if $(s \cdot e \cdot s') \sqsubset \alpha$, then $R_s = \{e\}$ and $L_{s'} = \{e\}$.*

In order to illustrate Theorem 22, we consider a few examples:

▶ **Example 23.** Let,

$$\begin{aligned}
\alpha &:= 1 \cdot 2 \cdot \mathbf{3} \cdot \mathbf{4} \cdot 5 \cdot 6 \cdot \mathbf{4} \cdot \mathbf{3} \cdot 7 \cdot 8 \cdot \mathbf{3} \cdot 9 \cdot 10,\\
\beta &:= 1 \cdot 2 \cdot \mathbf{4} \cdot 5 \cdot 6 \cdot \mathbf{3} \cdot \mathbf{4} \cdot 7 \cdot 8 \cdot \mathbf{3} \cdot 9 \cdot 10 \cdot \mathbf{4} \cdot \mathbf{3} \cdot 11 \cdot 12,\\
\gamma &:= 1 \cdot 2 \cdot \mathbf{3} \cdot \mathbf{4} \cdot 5 \cdot 6 \cdot \mathbf{7} \cdot \mathbf{8} \cdot 9 \cdot 10 \cdot \mathbf{4} \cdot \mathbf{3} \cdot 11 \cdot 12 \cdot \mathbf{8} \cdot \mathbf{7} \cdot 13 \cdot 14.
\end{aligned}$$

Then, according to Definition 6, E_α, E_β and E_γ are nonempty (the respective variables are typeset in bold face). However, since the patterns satisfy Theorem 22, there is no weakly unambiguous morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$ with respect to them if $|\Sigma| = 2$.

Theorem 22 and Example 23 directly imply the insight mentioned above that Theorem 13 does not hold for binary alphabets Σ:

▶ **Corollary 24.** *Let Σ be an alphabet with $|\Sigma| = 2$. There is an $\alpha \in \mathbb{N}^+$ such that E_α is not empty and there is no weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$ with respect to α.*

In contrast to the previous theorems, the following result features a *sufficient* condition on the *existence* of weakly unambiguous length-increasing morphisms $\sigma : \mathbb{N}^+ \to \Sigma^+$, $|\Sigma| = 2$, with respect to a given pattern. This phenomenon partly depends on the question of whether we can avoid short squares in the morphic image.

▶ **Theorem 25.** *Let $\alpha \in \mathbb{N}^+$, and let Σ be an alphabet, $|\Sigma| = 2$. Also, assume that*

- *$i \cdot e \cdot e' \sqsubset \alpha$ and $i \cdot e' \cdot e \sqsubset \alpha$, or*
- *$e \cdot e' \cdot i \sqsubset \alpha$ and $e' \cdot e \cdot i \sqsubset \alpha$,*

with $e, e' \in E_\alpha$ and $i \in \mathrm{var}(\alpha)$. If a morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$ satisfies

- *$|\sigma(e)| = 2$ and $|\sigma(e')| = 2$,*
- *for every $j \in \mathrm{var}(\alpha) \setminus \{e, e'\}$, $|\sigma(j)| = 1$, and*
- *there is no $x \in \Sigma$ with $x^2 \sqsubseteq \sigma(\alpha)$,*

then σ is weakly unambiguous with respect to α.

The main difference between Theorem 25 and Theorem 22 is that those patterns α being examined in Theorem 25 do not satisfy Condition 3 of Theorem 22. Thus, the two theorems demonstrate what subtleties in the structure of a pattern can determine whether or not it has a weakly unambiguous morphism with a binary target alphabet.

In order to illustrate Theorem 25, we now give some examples. In contrast to Example 23, the factors $3 \cdot 4$ and $4 \cdot 3$ of the patterns in the following example have an identical right neighbour or an identical left neighbour.

▶ **Example 26.** Let $\sigma : \mathbb{N}^+ \to \{a, b\}^+$ be a morphism. We define the morphism σ for the following patterns α (where the factors featured by Theorem 25 are typeset in bold face) as follows:

- $\alpha = 1 \cdot 2 \cdot \mathbf{5} \cdot \mathbf{3} \cdot \mathbf{4} \cdot 6 \cdot 7 \cdot 8 \cdot \mathbf{5} \cdot \mathbf{4} \cdot \mathbf{3} \cdot 9 \cdot 10$.
  σ is defined by $\sigma(1) := a$, $\sigma(2) := b$, $\sigma(5) := a$, $\sigma(3) := ba$, $\sigma(4) := ba$, $\sigma(6) := b$, $\sigma(7) := a$, $\sigma(8) := b$, $\sigma(9) := b$ and $\sigma(10) := a$.
- $\alpha = 1 \cdot 2 \cdot \mathbf{3} \cdot \mathbf{4} \cdot \mathbf{5} \cdot 6 \cdot 7 \cdot \mathbf{4} \cdot \mathbf{3} \cdot \mathbf{5} \cdot 8 \cdot 9$.
  σ is defined by $\sigma(1) := a$, $\sigma(2) := b$, $\sigma(3) := ab$, $\sigma(4) := ab$, $\sigma(5) := b$, $\sigma(6) := a$, $\sigma(7) := b$, $\sigma(8) := b$ and $\sigma(9) := a$.
- $\alpha = 1 \cdot 2 \cdot \mathbf{3} \cdot \mathbf{4} \cdot 5 \cdot 6 \cdot 7 \cdot \mathbf{8} \cdot \mathbf{3} \cdot \mathbf{4} \cdot 9 \cdot 10 \cdot 11 \cdot \mathbf{8} \cdot \mathbf{4} \cdot \mathbf{3} \cdot 12 \cdot 13$.
  σ is defined by $\sigma(1) := b$, $\sigma(2) := a$, $\sigma(3) := ba$, $\sigma(4) := ba$, $\sigma(5) := b$, $\sigma(6) := a$, $\sigma(7) := b$, $\sigma(8) := a$, $\sigma(9) := b$, $\sigma(10) := a$, $\sigma(11) := b$, $\sigma(12) := b$ and $\sigma(13) := a$.

With reference to Theorem 25, it can be easily verified that, in all above cases, σ is length-increasing and weakly unambiguous with respect to α.

The patterns in Example 26 further illustrate that the converse of Theorem 20 does not hold true. More precisely, although for every pattern α in this example, for every $e \in E_\alpha$ there exists an $e' \in E_\alpha$, $e' \neq e$, such that $e \cdot e'$ and $e' \cdot e$ are factors of α, there is a weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \to \{a, b\}^+$ with respect to α.

Due to Theorems 22 and 25, we expect that it is an extremely challenging task to find an equivalent to the characterisation in Theorem 13 for the binary case. From our understanding of the matter, we can therefore merely give the following conjecture on the decidability of Problem 2 for binary target alphabets.

▶ **Conjecture 27.** *Let $\alpha \in \mathbb{N}^+$ with $|\alpha| > 1$, and let $|\Sigma| := 2$. The problem of whether there is a weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$ with respect to α is decidable by testing a finite number of morphisms.*

The above conjecture is based on the fact that according to the Corollary 19, any weakly unambiguous length-increasing morphism with respect to a pattern α must not be length-increasing for the variables in S_α. On the other hand, increasing the length of the morphic images of the variables in E_α under a morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$, $|\Sigma| = 2$, seems to increase the chance of the existence of a morphism $\tau : \mathbb{N}^+ \to \Sigma^+$ satisfying $\tau(\alpha) = \sigma(\alpha)$ and, for some $i \in \text{var}(\alpha)$, $\tau(i) \neq \sigma(i)$. Consequently, we believe that if all morphisms σ with, for every $e \in E_\alpha$ and an $x \in \mathbb{N}$, $|\sigma(e)| \leq x$ are not weakly unambiguous with respect to α, then there does not exist a weakly unambiguous morphism σ with $|\sigma(e)| > x$ for some $e \in E_\alpha$, either. For all patterns, we expect a value of $x = 2$ to be a sufficiently large bound for the morphisms to be tested.

6 Weakly unambiguous morphisms with $|\Sigma| = 1$

It is not surprising that most of our considerations in the previous sections are not applicable to morphisms with a unary target alphabet. On the other hand, Corollary 18 and Corollary 19 also hold for this special case, i. e., for any pattern α, every weakly unambiguous morphism must map the variables in S_α to words of length 1, which implies that such a morphism can only be length-increasing if E_α is not empty. Incorporating these observations, we now consider an example.

▶ **Example 28.** Let $\alpha_1 := 1 \cdot 2 \cdot 3 \cdot 4 \cdot 1 \cdot 2 \cdot 3$. Consequently, $E_{\alpha_1} = \{4\}$. We define a morphism $\sigma : \mathbb{N}^+ \to \{a\}^+$ by $\sigma(4) := aa$ and $\sigma(i) := a$, $i \in \mathbb{N} \setminus \{4\}$. It can be easily verified that σ is weakly unambiguous with respect to α_1. Now let $\alpha_2 := 1 \cdot 2 \cdot 3 \cdot 4 \cdot 1 \cdot 2 \cdot 3 \cdot 5 \cdot 6$. As a result, $E_{\alpha_2} = \{4\}$. If we now consider the morphism τ, given by $\tau(4) := a$, $\tau(5) := aa$ and $\tau(i) := \sigma(i)$, $i \in \mathbb{N} \setminus \{4, 5\}$, then we may conclude $\tau(\alpha_2) = \sigma(\alpha_2)$. Thus, σ is not weakly unambiguous with respect to α_2.

Quite obviously, the fact that σ is unambiguous with respect to α_1 and ambiguous with respect to α_2 is due to 4 being the only variable in α_1 that has a single occurrence, whereas α_2 also has single occurrences of the variables 5 and 6. This aspect is reflected by the following characterisation that completely solves Problem 2 for morphisms with unary target alphabets.

▶ **Theorem 29.** *Let $\alpha \in \mathbb{N}^+$, $\text{var}(\alpha) = \{1, 2, 3, ..., n\}$. There is no weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \to \{a\}^+$ with respect to α if and only if, for every $i \in \text{var}(\alpha)$, there exist $n_1, n_2, ..., n_n \in \mathbb{N} \cup \{0\}$, such that*

$$|\alpha|_i = n_1|\alpha|_1 + n_2|\alpha|_2 + [...] + n_{i-1}|\alpha|_{i-1} + n_{i+1}|\alpha|_{i+1} + [...] + n_n|\alpha|_n.$$

Hence, we are able to provide a result on unary alphabets that is as strong as our result in Theorem 13 on ternary and larger alphabets. However, while Theorem 13 needs to consider the order of variables in the patterns, it is evident that Theorem 29 can exclusively refer to their number of occurrences.

7 Conclusion

In this paper, we have demonstrated that there is a weakly unambiguous length-increasing morphism $\sigma : \mathbb{N}^+ \to \Sigma^+$, $|\Sigma| \geq 3$, with respect to $\alpha \in \mathbb{N}^+$ if and only if E_α is not empty, where

$E_\alpha \subseteq \text{var}(\alpha)$ consists of those variables that have special, namely *illoyal* neighbour variables. We have demonstrated that this condition is not characteristic, but only necessary for the case $|\Sigma| = 2$, which leads to an interesting difference between binary and all other target alphabets Σ. We have not been able to characterise the existence of weakly unambiguous length-increasing morphisms with binary target alphabets, but we have found strong conditions that are either sufficient or necessary. Finally, for $|\Sigma| = 1$, we have been able to demonstrate that the existence of weakly unambiguous length-increasing morphisms $\sigma : \mathbb{N}^+ \to \Sigma^+$ solely depends on particular equations that the numbers of occurrences of the variables in the corresponding pattern need to satisfy.

Acknowledgements The authors are indebted to the anonymous referees, whose careful remarks and suggestions helped to correct Theorem 22 and the proof of Theorem 20.

References

1 J. Berstel and D. Perrin. *Theory of Codes*. Academic Press, Orlando, 1985.

2 D.D. Freydenberger and D. Reidenbach. The unambiguity of segmented morphisms. *Discrete Applied Mathematics*, 157:3055–3068, 2009.

3 D.D. Freydenberger, D. Reidenbach, and J.C. Schneider. Unambiguous morphic images of strings. *International Journal of Foundations of Computer Science*, 17:601–628, 2006.

4 D. Hamm and J. Shallit. Characterization of finite and one-sided infinite fixed points of morphisms on free monoids. Technical Report CS-99-17, Dep. of Computer Science, University of Waterloo, 1999. `http://www.cs.uwaterloo.ca/~shallit/papers.html`.

5 T. Harju and J. Karhumäki. Morphisms. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 7, pages 439–510. Springer, 1997.

6 S. Holub. Polynomial-time algorithm for fixed points of nontrivial morphisms. *Discrete Mathematics*, 309:5069–5076, 2009.

7 A. Mateescu and A. Salomaa. Patterns. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1, chapter 4.6, pages 230–242. Springer, 1997.

8 D. Reidenbach. A non-learnable class of E-pattern languages. *Theoretical Computer Science*, 350:91–102, 2006.

9 D. Reidenbach. An examination of Ohlebusch and Ukkonen's conjecture on the equivalence problem for E-pattern languages. *Journal of Automata, Languages and Combinatorics*, 12:407–426, 2007.

10 D. Reidenbach. Discontinuities in pattern inference. *Theoretical Computer Science*, 397:166–193, 2008.

11 D. Reidenbach and J.C. Schneider. Restricted ambiguity of erasing morphisms. In *Proc. 14th International Conference on Developments in Language Theory, DLT 2010*, volume 6224 of *Lecture Notes in Computer Science*, pages 387–398, 2010.

12 J.C. Schneider. Unambiguous erasing morphisms in free monoids. *RAIRO Informatique théoretique et Applications*, 44:193–208, 2010.

On Minimal Sturmian Partial Words*

Francine Blanchet-Sadri[1] and John Lensmire[2]

1 Department of Computer Science, University of North Carolina
P.O. Box 26170, Greensboro, NC 27402–6170, USA
blanchet@uncg.edu

2 Department of Mathematics, UCLA
Box 951555, Los Angeles, CA 90095, USA
jlensmire@gmail.com

Abstract

Partial words, which are sequences that may have some undefined positions called holes, can be viewed as sequences over an extended alphabet $A_\diamond = A \cup \{\diamond\}$, where $\diamond$ stands for a hole and matches (or is *compatible* with) every letter in A. The *subword complexity* of a partial word w, denoted by $p_w(n)$, is the number of distinct full words (those without holes) over the alphabet that are compatible with factors of length n of w. A function $f : \mathbb{N} \to \mathbb{N}$ is (k, h)-*feasible* if for each integer $N \geq 1$, there exists a k-ary partial word w with h holes such that $p_w(n) = f(n)$ for all n, $1 \leq n \leq N$. We show that when dealing with feasibility in the context of finite binary partial words, the only linear functions that need investigation are $f(n) = n + 1$ and $f(n) = 2n$. It turns out that both are $(2, h)$-feasible for all non-negative integers h. We classify all minimal partial words with h holes of order N with respect to $f(n) = n + 1$, called Sturmian, computing their lengths as well as their numbers, except when $h = 0$ in which case we describe an algorithm that generates all minimal Sturmian full words. We show that up to reversal and complement, any minimal Sturmian partial word with one hole is of the form $a^i\diamond a^j ba^l$, where i, j, l are integers satisfying some restrictions, that all minimal Sturmian partial words with two holes are one-periodic, and that up to complement, $\diamond(a^{N-1}\diamond)^{h-1}$ is the only minimal Sturmian partial word with $h \geq 3$ holes. Finally, we give upper bounds on the lengths of minimal partial words with respect to $f(n) = 2n$, which are tight for $h = 0, 1$ or 2.

1998 ACM Subject Classification F.2.2; F.4.3

Keywords and phrases Automata and formal languages; Combinatorics on words; Graph theory; Subword complexity; Feasible functions; Partial words; Sturmian words.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.225

1 Introduction

Let A be a k-letter alphabet and w be a finite or infinite word over A. A *subword* or *factor* of w is a block of consecutive letters of w. The *subword complexity* of w is the function which assigns to each integer n, the number, $p_w(n)$, of distinct subwords of length n of w. The subword complexity of finite and infinite words has become an important topic in combinatorics on words. Application areas include dynamical systems, ergodic theory, and theoretical computer science. Infinite words achieving various subword complexities have been widely studied: $p_w(n) = n + 1$ [13, 11], $p_w(n) = 2n$ [14], $p_w(n) = 2n + 1$ [4], to name a few (see Allouche [2] and Ferenczi [9] for some surveys). Chapter 10 of Allouche and Shallit's

* This material is based upon work supported by the National Science Foundation under Grant No. DMS–0754154. The Department of Defense is also gratefully acknowledged.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 225–236
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

book [3] provides a good overview for subword complexity of finite and infinite words. Our focus in this paper is on finite words.

Motivated by molecular biology of nucleic acids, Berstel and Boasson introduced partial words which are finite sequences that may have some undefined positions called holes (a (full) word is just a partial word without holes) [5]. Partial words can be viewed as sequences over an extended alphabet $A_\diamond = A \cup \{\diamond\}$, where $\diamond \notin A$ stands for a hole. Here $\diamond$ matches (or is *compatible* with) every letter in the alphabet. In this context, $p_w(n)$ is the number of distinct full words over the alphabet that are compatible with factors of length n of the partial word w (if $A = \{a, b\}$ and $w = a\diamond abaa$, then $p_w(3) = 5$ since aaa, aab, aba, baa and bab match factors of length 3 of w). Manea and Tiseanu showed that computing the subword complexity of partial words is a “hard” problem [12].

In this paper, we investigate minimal partial words with given subword complexity. This was done for a particular case of full words in [16]. There, it was shown that the minimal length of a word w such that $p_w(n) = F_{n+2}$ for all n, $1 \leq n \leq N$ is $F_N + F_{N+2}$, where $(F_n)_{n\geq 1}$ is the Fibonacci sequence and N is a positive integer, and an algorithm was given for generating such minimal words for each $N \geq 1$.

A function $f : \mathbb{N} \to \mathbb{N}$ is called (k, h)-*feasible* if for each integer $N \geq 1$, there exists a k-ary partial word w with h holes such that $p_w(n) = f(n)$ for all n, $1 \leq n \leq N$. In this case, w is an f-complex k-ary partial word with h holes of order N. Note that this is equivalent to saying there exists an integer N_0 such that for each $N \geq N_0$, there exists a k-ary partial word w with h holes such that $p_w(n) = f(n)$ for all n, $1 \leq n \leq N$. If f is a feasible function, it is immediate that f is non-decreasing and let us denote the length of a shortest f-complex k-ary partial word w with h holes of order N (called *minimal*) by $L_k(f, N, h)$. Similarly, denote the number of such minimal partial words by $N_k(f, N, h)$.

First, let us consider functions of the form $f(n) = k^n$, where k is the alphabet size. When we restrict our attention to the case of $h = 0$, a k-ary de Bruijn sequence of order N is a full word over a k-letter alphabet A where each of the k^n full words of length n over A appears as a factor exactly once. It is well known that $L_k(k^n, N, 0) = k^n + n - 1$. Moreover, $N_k(k^n, N, 0) = k!^{k^{n-1}}$, and these sequences can be efficiently generated by constructing Eulerian cycles in corresponding de Bruijn directed graphs. The technical report of de Bruijn provides a history on the existence of these sequences [8]. De Bruijn graphs find applications, in particular, in genome rearrangements [1], etc.

In [7], the case of $h > 0$ was initiated. For positive integers N, h and k, Blanchet-Sadri et al. introduced the concept of a de Bruijn partial word of order N with h holes over an alphabet A of size k, as being a partial word w with h holes over A of minimal length with the property that $p_w(n) = k^n$. There, the authors gave lower and upper bounds on $L_k(k^n, N, h)$, and showed that their bounds are tight when $h = 1$ and $k \in \{2, 3\}$. They provided an algorithm to construct 2-ary de Bruijn partial words with one hole of order N. Finally, they showed how to compute $N_2(2^n, N, 1)$ by adapting the so called BEST theorem that counts the number of Eulerian cycles in directed graphs [15].

Now, let us look at constant functions over the binary alphabet $\{a, b\}$. Note that $f \equiv 1$ is $(2, 0)$-feasible, and that a^N and b^N are the only minimal f-complex full words of order N (so that $L_2(1, N, 0) = N$ and $N_2(1, N, 0) = 2$). Furthermore, $f \equiv 1$ is not $(2, h)$-feasible for any $h \geq 1$, as any $\diamond$ in a partial word w implies that $2 = p_w(1) = f(1)$. Note also that $f \equiv 2$ is $(2, 0)$- and $(2, 1)$-feasible, but not $(2, h)$-feasible for $h \geq 2$. To see this, words of the form ab^N and $\diamond a^{N-1}$ show that f is $(2, 0)$- and $(2, 1)$-feasible respectively. Furthermore, these words are minimal and unique up to reverse and complement. Thus, $L_2(2, N, 0) = N + 1$, $L_2(2, N, 1) = N$, and $N_2(2, N, 0) = N_2(2, N, 1) = 4$. Now suppose that a word w has at least two holes. If w has two consecutive holes, note that $p_w(2) = 4$. If the holes are spread out, e.g. both $\diamond c$ and $d\diamond$ are factors of w for some letters $c, d \in \{a, b\}$, then $p_w(2) \geq 3$.

In this paper, let us investigate linear functions for binary partial words. It is obvious that if $f(1) = 1$, then $f \equiv 1$. Thus, when characterizing linear functions f, we only need to look at the case when $f(1) = 2$, that is, $f(n) = pn + q$ for integers p, q such that $p + q = 2$ and $p > 0$. Note that if $p > 2$, then $f(2) > 4$. Thus, the only linear options are $f(n) = n + 1$ or $f(n) = 2n$. The contents of our paper is as follows: In Section 2, we review some basics on partial words. We also give a bound on the subword complexity of any binary partial word with h holes. In Section 3, we show that the linear function $f(n) = n + 1$ is $(2, h)$-feasible for all non-negative integers h, and we consider $(n + 1)$-complex partial words referred to as *Sturmian*. We classify all minimal Sturmian partial words with h holes of order N, computing the exact length $L_2(n + 1, N, h)$ as well as the exact number $N_2(n + 1, N, h)$, except for $N_2(n + 1, N, 0)$. Instead of computing the latter, we describe an algorithm that generates all Sturmian full words of order N. We show that any minimal Sturmian partial word with one hole is of the form $a^i\diamond a^j ba^l$ (up to reversal and complement), where i, j, l are integers satisfying some restrictions, that all minimal Sturmian partial words with two holes are one-periodic, and that up to complement, $\diamond(a^{N-1}\diamond)^{h-1}$ is the only minimal Sturmian partial word with $h \geq 3$ holes. Finally in Section 4, we prove that the linear function $f(n) = 2n$ is also $(2, h)$-feasible for all non-negative integers h, and we conclude with some results on $2n$-complex partial words.

2 Preliminaries

We recall some basic terminology and notation on partial words that are useful throughout the paper. For more background, we refer the reader to [6].

Let A be a nonempty finite set of symbols called an *alphabet*. Each element $a \in A$ is called a *letter*. A *partial word* over A is a finite sequence of symbols from the alphabet enlarged with the *hole* symbol, $A_\diamond = A \cup \{\diamond\}$, where a *(full) word* is a partial word which does not contain any $\diamond$'s. The *length* of a partial word u is denoted by $|u|$ and represents the number of symbols in u. The *empty word* has length zero and is denoted by ε. If $\mathcal{S}$ is a set of partial words, $\|\mathcal{S}\|$ denotes its cardinality.

We denote by $u(i)$ the symbol at position i of the partial word u, the labelling of the positions starting at 0. Position i in u is in the *domain* of u, denoted by $D(u)$, if $u(i) \in A$. Otherwise if $u(i) = \diamond$, position i belongs to the *set of holes* of u. A positive integer p is called a *period* of a partial word u if $u(i) = u(j)$ whenever $i, j \in D(u)$ and $i \equiv j \mod p$. In such a case, we call u *p-periodic*. The powers of u are defined recursively by $u^0 = \varepsilon$ and for $n \geq 1$, $u^n = uu^{n-1}$.

A *completion* of a partial word w over A is a full word $\hat{w}$ constructed by filling in the holes of w with letters from A. If u and v are two partial words of equal length, then u and v are *compatible*, denoted by $u \uparrow v$, if $u(i) = v(i)$ whenever $i \in D(u) \cap D(v)$, that is there exist completions $\hat{u}, \hat{v}$ such that $\hat{u} = \hat{v}$.

A partial word u is a *factor* of a partial word v if there exist partial words x, y such that $v = xuy$. We adopt the notation $v[i..j)$ to denote the factor $v(i) \cdots v(j - 1)$ of v. Here u is a *prefix* of v if $x = \varepsilon$ and a *suffix* of v if $y = \varepsilon$. A full word u is a *subword* of a partial word w if $u \uparrow v$ for some factor v of w. Informally, u is a subword of w if there is some completion $\hat{w}$ such that u is a factor of $\hat{w}$. Note that subwords in this paper are always full. We let $\text{Sub}_w(n)$ denote the set of all subwords of w of length n, and we let $\text{Sub}(w) = \bigcup_{0 \leq n \leq |w|} \text{Sub}_w(n)$, the set of all subwords of w. Note that if $\hat{w}$ is a completion of w, then $p_{\hat{w}}(n) \leq p_w(n)$, since

$\mathrm{Sub}_{\hat{w}}(n) \subset \mathrm{Sub}_w(n)$.

We end this section by giving a bound on the subword complexity of any binary partial word w. Let $n \leq |w|$ be a positive integer. A factor u of length n of w is repeated, if there exist integers $i \neq j$ such that $u = w[i..i+n) = w[j..j+n)$. Similarly, a subword u of length n of w is repeated, if there exist integers $i \neq j$ such that $u \uparrow w[i..i+n)$ and $u \uparrow w[j..j+n)$. Note that repeated factors imply repeated subwords, but the converse does not hold in general.

▶ Proposition 1. Let w be a partial word with h holes over a binary alphabet. For index $i = 0, \dots, h$ and positive integer $n \leq |w|$, let $\mathcal{F}_i(w,n)$ denote the multiset containing the factors of w of length n with exactly i holes. Then

$$\sum_{i=0}^{h} \|\mathcal{F}_i(w,n)\| = |w| - n + 1 \tag{1}$$

$$p_w(n) \leq \sum_{i=0}^{h} 2^i \|\mathcal{F}_i(w,n)\| \tag{2}$$

with equality holding in (2) if and only if w has no repeated subwords of length n. The following zero-hole and one-hole bounds hold:

1. Let $h = 0$. For $n \leq |w|$, we have $p_w(n) \leq |w| - n + 1$, with equality holding if and only if w has no repeated subwords of length n.
2. Let $h = 1$ and $n \leq |w|$. If $|w| \leq 2n-1$, then $p_w(n) \leq 2(|w|-n+1)$. Else, $p_w(n) \leq |w|+1$. In both cases, equality holds if and only if w has no repeated subwords of length n.

Proof. For Statement (2), Inequality (2) implies that $p_w(n) \leq \|\mathcal{F}_0(w,n)\| + 2\|\mathcal{F}_1(w,n)\|$ with equality holding if and only if w contains no repeated subwords of length n. First suppose that $|w| \leq 2n-1$. In this case, it is possible that w satisfies $\mathcal{F}_0(w,n) = \emptyset$. Note that since Equality (1) holds, this situation maximizes the subword complexity. Therefore, $p_w(n) \leq 2\|\mathcal{F}_1(w,n)\| = 2(|w|-n+1)$. Now suppose that $|w| > 2n-1$. We have $\|\mathcal{F}_1(w,n)\| \leq n$. If $\|\mathcal{F}_1(w,n)\| = n$, then $\|\mathcal{F}_0(w,n)\| = |w| - 2n + 1$. Thus, $p_w(n) \leq |w| - 2n + 1 + 2n = |w| + 1$ as desired. ◀

3 Sturmian partial words

In this section, we investigate Sturmian partial words. Recall that a finite partial word w is called Sturmian of order N if $p_w(n) = n+1$ for all n, $1 \leq n \leq N$. We will fill out Table 1, whose first three columns show that for $h \geq 0$, $f(n) = n+1$ is $(2,h)$-feasible.

▶ Remark. Note that the lengths of the words in the third column of Table 1 give upper bounds on $L_2(n+1,N,h)$, listed in the fourth column. For $N \geq 1$, let $w = a^N b^N$. By Proposition 1(1), a word z must have length $l \geq 2N$ to satisfy $p_z(N) \geq N+1$. Thus, w is a minimal $(n+1)$-complex partial word of order N, and so $L_2(n+1,N,0) = 2N$.

Now for $N \geq 6$, let $w = a^{\lfloor N/2 \rfloor} \diamond a^{\lfloor N/2 \rfloor} b a^{\lceil (N-4)/2 \rceil}$. By Table 1, w is an $(n+1)$-complex partial word of order N with $|w| = \frac{3N}{2}$ when N is even, and $|w| = \frac{3N}{2} - \frac{1}{2}$ when N is odd. By Proposition 1(2), a word z with one hole must have length $l \geq \frac{3N}{2} - \frac{1}{2}$ to satisfy $p_z(N) \geq N+1$, implying that w is minimal, and so $L_2(n+1,N,1)$ is as shown in the table.

As is proved later, the other upper bounds also turn out to be lower bounds.

Table 1 Sturmian partial words with h holes of order N

h	N	partial word	$L_2(n+1,N,h)$	$N_2(n+1,N,h)$
0	≥ 1	$a^N b^N$	$2N$	
1	≥ 6	$a^{\lfloor N/2 \rfloor}\diamond a^{\lfloor N/2 \rfloor} b a^{\lceil (N-4)/2 \rceil}$	$\frac{3N}{2}$ if N is even $\frac{3N}{2}-\frac{1}{2}$ if N is odd	12 if N is even 4 if N is odd
2	≥ 12	$a^{\lfloor (N-6)/2 \rfloor}\diamond a^{N-5}\diamond a^{\lceil (N-6)/2 \rceil}$	$2N-9$	$2N-22$
≥ 3	$\geq h+1$	$\diamond(a^{N-1}\diamond)^{h-1}$	$N(h-1)+1$	2

3.1 The case of $h = 0$

The aim of this section is to provide an algorithm that generates all minimal Sturmian full words. In constructing them, some graph theory is useful (the reader is refereed to [10] for more information).

Let $G = (V, E)$ be a directed graph. The *line digraph* of G, denoted by $L(G)$, is the graph $G' = (V', E')$ where $V' = E$, and for all $v_1', v_2' \in V'$, $(v_1', v_2') \in E'$ if $v_1' = (v_1, v_2)$ and $v_2' = (v_2, v_3)$ for some $v_1, v_2, v_3 \in V$. Combining ideas from de Bruijn and Rauzy graphs, we define a labelled directed graph $G_{\mathcal{S}} = (V, E)$ on a set $\mathcal{S}$ of words of length n as follows: V consists of the set of factors of length $n-1$ of words in $\mathcal{S}$ and E consists of the set of edges (x, x') for which there exists $y \in \mathcal{S}$ such that x is a prefix of y and x' is a suffix of y (such edges are labelled by y). This definition provides us with a natural correspondence between graphs and words.

▶ **Lemma 3.1.** *Given a set $\mathcal{S}$ consisting of words of length n, there exists a word w such that $Sub_w(n) = \mathcal{S}$ if and only if $G_{\mathcal{S}} = (V, E)$ has a path that includes all of the edges of $G_{\mathcal{S}}$. If such a path p exists, then there exists a word w of length $|p| + n - 1$ with $Sub_w(n) = \mathcal{S}$. Furthermore, $Sub_w(n-1) \supset V$.*

The following properties of a directed graph $G = (V, E)$ are well known and are useful throughout this section. The notation $\texttt{ideg}(v)$ refers to the in-degree of vertex v, $\texttt{odeg}(v)$ to its out-degree, and $(\texttt{ideg}(v), \texttt{odeg}(v))$ to its degree.

1. The size of the line digraph $L(G) = (V', E')$ of G is $|V'| = |E|$ and $|E'| = \sum_{v \in V} \texttt{ideg}(v) \times \texttt{odeg}(v)$.
2. The graph G has an Eulerian circuit if and only if G is strongly connected and for every vertex $v \in V$, $\texttt{ideg}(v) = \texttt{odeg}(v)$.
3. If $x, y \in V$ are such that $\texttt{odeg}(x) = \texttt{ideg}(x) + 1$ and $\texttt{ideg}(y) = \texttt{odeg}(y) + 1$, then G has an (x, y)-Eulerian path (or an Eulerian path from x to y) if and only if G is weakly connected and for every vertex $v \in V \setminus \{x, y\}$, $\texttt{ideg}(v) = \texttt{odeg}(v)$.

We call a directed graph G *Sturmian of order* n if G has n vertices, $n+1$ edges, and contains an Eulerian path. The graph G is *Sturmian Type I or II* if G has degree sequence $(2,2), (1,1), \ldots, (1,1)$ or $(2,1), (1,2), (1,1), \ldots, (1,1)$ respectively.

▶ Proposition 2. **1.** Suppose that $G = (V, E)$ is Sturmian Type II of order n. Then $L(G)$ is Sturmian of order $n+1$.

2. Suppose that $G = (V, E)$ is Sturmian Type I of order n. Then it is possible to remove one edge from $L(G)$ to get G', where G' is Sturmian of order $n+1$. Furthermore, it is impossible to remove an edge from $L(G)$ to get a graph G' such that G' contains a path that contains all of the edges of G' and G' is not Sturmian.

Proof. For Statement (2), note that $L(G)$ has $n+1$ vertices and $n+3$ edges. Since G contains an Eulerian path, $L(G)$ has a Hamiltonian path, and thus is weakly connected. Thus, we are left to show that we can remove one edge from $L(G)$ to get a graph G' that is still weakly connected and contains an Eulerian path. The graph G being Sturmian Type I, there is a distinct vertex v that has degree $(2,2)$. Label the edges in and out of v as i_1, i_2 and o_1, o_2 respectively. Note that all the vertices in $L(G)$ not in $S = \{i_1, i_2, o_1, o_2\}$ have degree $(1,1)$. Two cases remain which are illustrated in Figure 1: Case (i) where each member in S is distinct, and Case (ii) where $i_2 = o_2$.

For Case (i), i_1, i_2 have degree $(1,2)$ while o_1, o_2 have degree $(2,1)$. Note that there are edges from i_j to o_l for each $j, l \in \{1,2\}$. Remove the edge (i_2, o_2) to get a graph G'. Note that G' is still weakly connected. Furthermore, in G', i_1 has degree $(1,2)$, o_1 has degree $(2,1)$, and all other vertices have degree $(1,1)$. Thus, G' has an Eulerian path and is Sturmian Type II. Note that removing any of the edges (i_j, o_l) can be handled similarly. Further, note that removing any other edge from $L(G)$ results in a graph that no longer has a path containing all the edges. ◀

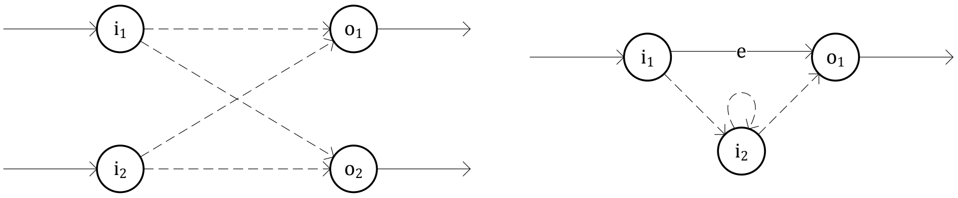


Figure 1 Part of $L(G)$ in Proposition 2(2): Left: Case (i); Right: Case (ii).

We are now ready to present an algorithm (similar to one used by Rote in [14]) to generate minimal Sturmian full words. Note that Proposition 2 implies that the graph G' created in line 2, 6, or 8 is always Sturmian. Since G'_N has $N+1$ edges, Algorithm 1 generates a minimal Sturmian word.

Algorithm 1 Constructing a minimal Sturmian full word of order $N \geq 3$.

```
1: Create $G_2 = G_{\mathcal{S}}$, where $\mathcal{S} = \{aa, ab, ba, bb\}$
2: Create $G'_2$ by deleting an edge from $G_2$
3: for $i = 3$ to N do
4:     Build $G_i = L(G'_{i-1})$
5:     if $G_i$ has $i+2$ edges then
6:         Create $G'_i$ by deleting an edge from $G_i$ (so that $G'_i$ has $i+1$ edges), but ensure that
           $G'_i$ still contains an Eulerian path
7:     else
8:         Set $G'_i = G_i$
9: Find an Eulerian path $p$ in $G'_N$
10: return $p$
```

▶ **Theorem 3.2.** *Algorithm 1 generates all minimal Sturmian full words.*

Proof. Suppose that w is a minimal Sturmian full word of order N. Thus, Lemma 3.1 implies that there is a sequence of graphs $G_2, \ldots, G_N$ such that G_i has i vertices and $i+1$ edges.

Furthermore, G'_2 is a subgraph of $G_{\mathcal{S}}$, where $\mathcal{S} = \{aa, ab, ba, bb\}$, for $i = 2, \ldots, N-1$, G_{i+1} is a subgraph of $L(G_i)$, and for each G_i there exists a path containing all its edges. Thus, w can be generated by Algorithm 1 unless there exists some G_i that does not contain an Eulerian path. However, since G_{i+1} is a subgraph of $L(G_i)$, Proposition 2 ensures that G_{i+1} contains an Eulerian path. ◀

3.2 The case of $h = 1$

Recall the minimal Sturmian partial word $a^{\lfloor N/2 \rfloor} \diamond a^{\lfloor N/2 \rfloor} ba^{\lceil (N-4)/2 \rceil}$ of order $N \geq 6$ in Table 1, which has the form $a^i \diamond a^j ba^l$ for some i, j, l. We show that any minimal Sturmian partial word with one hole has a similar form. Note that since $N \geq 6$, any Sturmian partial word w of order N with one hole satisfies $N < |w|$ (otherwise, $N = |w|$, and we get $N + 1 = p_w(N) = 2$, a contradiction).

▶ **Lemma 3.3.** *1.* *For $N \geq 6$, any Sturmian partial word w of order N of the form $w = \diamond z$, where z is a full word, is not minimal.*

2. *Any Sturmian partial word w of order N of the form $w = a^i \diamond (a^j b)^m y$, where $i, j \geq 1$, $m \geq 2$, and y is a prefix of $a^j b$, is not minimal.*

Proof. For Statement (2), we first prove that $N \leq \min(s, t)$, where $s = i + j + 1$ and $t = (j+1)m + |y| + 1$. First suppose that $s \leq t$ and $N \geq s + 1$. Note that $\mathrm{Sub}_w(s+1) = \{a^s b, \ldots, a^{i+1} ba^j, a^i ba^j b, \ldots, b(a^j b)^{s/(j+1)}\}$. This implies that $p_w(s+1) = s - (i+1) + 1 + i + 1 = s + 1 < s + 2$, a contradiction.

Next suppose that $t < s$ (so that $t + 1 \leq i + j + 1$). If $N \geq t + 1$, then $\mathrm{Sub}_w(t+1) = \{a^{t+1}, a^t b, \ldots, ab(a^j b)^{(t-1)/(j+1)}\}$, so $p_w(t+1) = 1 + t - 1 + 1 = t + 1 < t + 2$, a contradiction. Hence, $N \leq \min(s, t)$ as claimed. Therefore, if w has order N, then $s, t \geq N$ or $i + j + 1, (j+1)m + |y| + 1 \geq N$. Thus, $|w| = i + 1 + (j+1)m + |y| = s - j - 1 + t$. For a fixed t, j takes on a maximum value when $m = 2$ and $|y| = 0$. Hence, $2(j+1) + 1 \leq t$ so that $j \leq \frac{t-3}{2}$ and $|w| = s - j - 1 + t \geq s - \frac{t-3}{2} - 1 + t = s + \frac{t}{2} + \frac{1}{2} \geq \frac{3N}{2} + \frac{1}{2}$. However, $L_2(n+1, N, 1) \leq \frac{3N}{2}$ from Remark 3, so w is not minimal. ◀

▶ **Theorem 3.4.** *Suppose w is a Sturmian partial word with one hole of order $N \geq 6$ with a factor $z = \diamond a^i b$, where $i \geq 1$. Then w contains no other b's or w is not minimal.*

Proof. Similarly to the above lemma, we use the fact (from Remark 3) that if $|w| > \frac{3N}{2}$ then w is not minimal. If w contains no other b's we are done. Otherwise, w contains a factor of the form $ba^j z$ or $za^j b$, for some $j \geq 1$. Note that if $j = 0$, w would contain all the four subwords of length 2, contrary to our assumption that w is Sturmian. First assume that $u = ba^j \diamond a^i b$ is a factor of w. Let $t = \min(i, j)$. Note that

$$\mathrm{Sub}_u(t+2) = \{a^{t+2}, ba^{t+1}, a^t ba, \ldots, aba^t, a^{t+1} b, ba^t b\}$$

has size $t + 4$, implying that $N \leq t + 1$. Thus, $|w| \geq |u| = i + j + 3 \geq 2t + 3 > 2t + 2 \geq 2N$, so w is not minimal. Next assume that $u = \diamond a^i ba^j b$ is a factor of w.

First, suppose that $i > j$. Thus,

$$\mathrm{Sub}_u(j+2) = \{a^{j+2}, ba^{j+1}, a^{j+1} b, aba^j, \ldots, a^j ba, ba^j b\}$$

has size $j + 4$ implying that $N \geq j + 1$ Similarly to the above, this implies that $|w| \geq 2N$ and w is not minimal. The case where $j > i + 1$ is handled similarly since $\mathrm{Sub}_u(i+2)$ is too large. Now, suppose that $i = j$. So $w = xuy = x \diamond a^i ba^i by$ for some full words x, y. Note that if x contains a b, it has already been shown that w is not minimal. Furthermore, if $x = \varepsilon$, then w is not minimal by Lemma 3.3(1). Therefore, $w = a^l \diamond (a^i b)^2 y$ for some

$l \geq 1$. Note that if $N < i+2$, then $|w| \geq 2N$ and w is not minimal. So suppose $N \geq i+2$. We have that $\mathrm{Sub}_w(i+2) \supset \{a^{i+2}, a^{i+1}b, a^iba, \ldots, aba^i, ba^ib\} = \mathcal{S}$. Since the latter set is of size $i+3$, w must avoid $\{a,b\}^{i+2} \setminus \mathcal{S}$. Thus, $w = a^l\diamond(a^ib)^my$ for some $m \geq 2$ and some prefix y of a^ib, and by Lemma 3.3(2), w is not minimal. Finally, suppose that $j = i+1$. So $w = xuy = x\diamond a^iba^{i+1}by$ for some full words x, y. Similarly to the above, we only need to consider the case when $w = a^l\diamond a^iba^{i+1}by$ for some $l \geq 1$. Note that $\mathrm{Sub}_w(i+2) \supset \{a^{i+2}, ba^{i+1}, a^iba, \ldots, aba^i, a^{i+1}b, ba^ib\}$, so $\|\mathrm{Sub}_w(i+2)\| \geq i+4$ and $N \leq i+1$. However, this implies that $|w| \geq 2N$, and w is not minimal. ◀

▶ **Corollary 3.5.** *For $N \geq 6$, any minimal Sturmian partial word with one hole is of the form $a^i\diamond a^jba^l$ for some i, j, l (up to reversal and complement).*

The next lemma gives some restrictions on the integers i, j, l.

▶ **Lemma 3.6.** *Let $w = a^i\diamond a^jba^l$ be a minimal Sturmian partial word with one hole of order $N \geq 6$. If N is odd, w has no repeated subwords of length N, and $i, j+l+1 < N$ (e.g. all factors of w of length N contain a hole). If N is even, exactly one of the following holds:*

- *w has exactly one subword of length N repeated exactly once, and $i, j+l+1 < N$.*
- *w has no repeated subwords of length N, and $i < N$, $j+l+1 = N$.*

Proof. Assume that N is odd. Thus, $|w| = \frac{3N}{2} - \frac{1}{2} \leq 2N-1$. From Proposition 1(2), $p_w(N) \leq 2(|w| - N + 1) = N+1$, and we have equality if and only if w has no repeated subwords of length N. Furthermore, the proof of Proposition 1(2) shows that each factor of w of length N contains a hole, and so $i, j+l+1 < N$.

Assume that N is even. Thus, $|w| = \frac{3N}{2} \leq 2N-1$ and $p_w(N) \leq 2(|w| - N + 1) = N+2$ from Proposition 1(2). More details follow. If $\|\mathcal{F}_0(w,N)\| \geq 2$, then $\|\mathcal{F}_1(w,N)\| \leq |w| - N - 1$ and $p_w(N) \leq \|\mathcal{F}_0(w,N)\| + 2\|\mathcal{F}_1(w,N)\| \leq N$, and so w is not Sturmian. If $\|\mathcal{F}_0(w,N)\| = 1$, then $\|\mathcal{F}_1(w,N)\| = |w| - N$ and $p_w(N) \leq \|\mathcal{F}_0(w,N)\| + 2\|\mathcal{F}_1(w,N)\| = N+1$, and equality holds if and only if no subword of length N repeats. This can only be the case when $i < N, j+l+1 = N$ (note that w has a^N as a repeated subword of length N when $i = N$, $j+l+1 < N$). If $\|\mathcal{F}_0(w,N)\| = 0$, then $\|\mathcal{F}_1(w,N)\| = |w| - N + 1$ and $p_w(N) \leq \|\mathcal{F}_0(w,N)\| + 2\|\mathcal{F}_1(w,N)\| \leq N+2$, and so $p_w(N) = N+1$ implies that exactly one subword must repeat exactly once. This can only be the case when $i, j+l+1 < N$. Again, the proof of Proposition 1(2) makes it evident that the two cases listed above are the only ones that lead to $p_w(N) = N+1$. ◀

The next lemma gives upper and lower bounds on j.

▶ **Lemma 3.7.** *Let $w = a^i\diamond a^jba^l$ be a minimal Sturmian partial word with one hole of order $N \geq 6$. Then $\lfloor\frac{N-1}{2}\rfloor \leq j \leq \lfloor\frac{N}{2}\rfloor$.*

Proof. To show the lower bound $j \geq \lfloor\frac{N-1}{2}\rfloor$, suppose that $j < \lfloor\frac{N-1}{2}\rfloor$. First suppose that $l \geq j+1$. Here $i, j \geq 1$. Since $N \geq 6$, we have that $N \geq j+2$. However, $\mathrm{Sub}_w(j+2) \supset \{a^{j+2}, a^{j+1}b, a^jba, \ldots, aba^j, ba^{j+1}, ba^jb\}$ so that $p_w(j+2) \geq j+4$. Thus, $l \leq j$.

Assume that N is even. Thus, $j = \frac{N}{2} - m$ for some $m \geq 2$. Noting that $|w| = \frac{3N}{2} = i+j+l+2$ we have that $i \geq \frac{N}{2} - 2 + 2m$. Thus, $i+j+1 \geq \frac{N}{2} - 2 + 2m + \frac{N}{2} - m + 1 \geq N + m - 1 \geq N+1$, with equality holding if and only if $l = j$. If $l = j$, both a^N and $a^{N-l-1}ba^l$ are repeated subwords of length N of w, contradicting Lemma 3.6. Similarly, $l < j$ implies that $i+j+1 \geq N+2$, meaning that a^N appears as a subword at least three times, again contradicting Lemma 3.6. ◀

The next theorem gives the classification of the one-hole minimal Sturmian words.

▶ **Theorem 3.8.** *Let $N \geq 6$.*

1. *If N is odd, then the only minimal Sturmian partial word with one hole of order N (up to reversal and complement) is $a^{N/2-1/2} \diamond a^{N/2-1/2} b a^{N/2-3/2}$, or equivalently $a^{\lfloor N/2 \rfloor} \diamond a^{\lfloor N/2 \rfloor} b a^{\lceil (N-4)/2 \rceil}$, and so $N_2(n+1, N, 1) = 4$.*
2. *If N is even, then the only minimal Sturmian partial words with one hole of order N (up to reversal and complement) are $a^{N/2} \diamond a^{N/2-1} b a^{N/2-1}$, $a^{N/2-1} \diamond a^{N/2} b a^{N/2-1}$, and $a^{N/2} \diamond a^{N/2} b a^{N/2-2} = a^{\lfloor N/2 \rfloor} \diamond a^{\lfloor N/2 \rfloor} b a^{\lceil (N-4)/2 \rceil}$, and so $N_2(n+1, N, 1) = 12$.*

Proof. Let w be a minimal Sturmian partial word with one hole. By Corollary 3.5, $w = a^i \diamond a^j b a^l$ for some i, j, l. For Statement (2), when N is even, $j = \frac{N}{2} - 1$ or $j = \frac{N}{2}$. Assume that $j = \frac{N}{2} - 1$. From Lemma 3.6 we have two cases to consider. Suppose $j + l + 1 = N$, so that $l = \frac{N}{2}$ and $i = \frac{N}{2} - 1$. Setting $t = \frac{N}{2} + 1$, we have that $t \leq N$ and that $\text{Sub}_w(t) = \{a^t, a^{t-1}b, a^{t-2}ba, \ldots, aba^{t-2}, ba^{t-1}, ba^{t-2}b\}$ is of size $t+2$, a contradiction. Thus, $i, j+l+1 < N$, and w can have at most one repeated subword of length N. Set $l = \frac{N}{2} - m$ for some $m \geq 1$, so that $i = \frac{N}{2} - 1 + m$. Further note that $a^i b a^l$ is a repeated subword of length N of w. We also have that $i + j + 1 = N - 1 + m$, so that if $m > 1$, a^N is also a repeated subword of length N, a contradiction. Therefore, $m = 1$ and $w = a^{N/2} \diamond a^{N/2-1} b a^{N/2-1}$. ◀

3.3 The case of $h = 2$

Recall from Table 1 that $a^{\lfloor (N-6)/2 \rfloor} \diamond a^{N-5} \diamond a^{\lceil (N-6)/2 \rceil}$ is a Sturmian partial word of order $N \geq 12$ of length $2N - 9$. We show that this form is minimal, and in fact all minimal Sturmian partial words with two holes are similar. The next proposition describes behavior between the holes.

▶ Proposition 3. Suppose that w is a Sturmian partial word of order N. Let z be a factor of w of the form $z = \diamond a_0 \cdots a_{l-1} \diamond$, where $a_0, \ldots, a_{l-1} \in \{a, b\}$. Then, $N < \frac{l}{2} + \frac{3}{2}$, or z is one-periodic, or $z = w = \diamond a^j b a^{n_1} b a^{n_2} b \cdots b a^{n_i} b a^j \diamond$ for some $i, j \geq 0$ and some $n_1, n_2, \ldots, n_i \in \{j, j+1\}$.

Proof. If $N < \frac{l}{2} + \frac{3}{2}$ we are done. Thus, assume $N \geq \frac{l}{2} + \frac{3}{2}$ throughout the rest of the proof. If $l < 2$ the statement is immediate. So assume that $l \geq 2$. Without loss of generality assume that $a_0 = a$. For j, $0 \leq j < \frac{l}{2}$, we show that either z avoids $ba^j b$ or $z = w = \diamond a^j b a^{n_1} b a^{n_2} b \cdots b a^{n_i} b a^j \diamond$ for some $i \geq 0$ and some $n_1, n_2, \ldots, n_i \in \{j, j+1\}$.

Assume first that $j = 0$. Suppose that $a_{l-1} = b$. Thus, $\diamond a$ and $b \diamond$ are factors of z, and $aa, ba, bb \in \text{Sub}_z(2)$. Since $p_z(2) = 3$, z must avoid ab. However, since $a_0 = a$ we have that $a_{l-1} = a$, a contradiction. Thus, $a_{l-1} = a$, and $aa, ab, ba \in \text{Sub}_z(2)$ implying that z avoids bb. Inductively, suppose that z avoids $bb, bab, \ldots, ba^{j-1}b$. This implies that $a_0 = \cdots = a_{j-1} = a_{l-1-j+1} = \cdots = a_{l-1} = a$. If z is one-periodic we are done, so suppose otherwise. Note that this also implies that $j < \frac{l}{2}$, else z would be one-periodic. Thus, $j + 2 \leq \frac{l}{2} + \frac{3}{2} \leq N$. Since z avoids $bb, \ldots, ba^{j-1}b$, we have that

$$\text{Sub}_z(j+2) \subset \{a^{j+2}, a^{j+1}b, a^j ba, \ldots, ba^{j+1}, ba^j b\} = \mathcal{S}$$

Note that $\|\mathcal{S}\| = j + 4$, so exactly one element of $\mathcal{S}$ must be avoided. Further, note that since z is not one-periodic, $\{a^{j+1}b, a^j ba, \ldots, ba^{j+1}\} \subset \text{Sub}_w(j+2)$. If z avoids $ba^j b$ we are done. Thus suppose that z avoids a^{j+2}. Thus, $z = \diamond a^j b a^{n_1} b a^{n_2} b \cdots b a^{n_i} b a^j \diamond$ for some $i \geq 0$ and $n_1, n_2, \ldots, n_i \in \{j, j+1\}$. Suppose that $z \neq w$, so we can write $w = xzy$ for some partial words x, y where at least one of $x, y \neq \varepsilon$. Without loss of generality assume that $y \neq \varepsilon$. Note that since $p_z(2) = p_w(2) = 3$, we have that w avoids bb. Therefore, $\diamond \neq y_0 \neq b$ so $y_0 = a$. However, this implies that a^{j+2} is a subword of w that is avoided by z, so that $p_w(j+2) > p_z(j+2) = j + 3$, a contradiction. Thus, both $x, y = \varepsilon$ and $w = z$. ◀

▶ **Corollary 3.9.** *Minimal Sturmian partial words of order $N \geq 12$ with two holes are one-periodic.*

It remains to restrict the placement of the holes.

▶ Proposition 4. Let $N \geq 12$. Any minimal Sturmian partial word w of order N with two holes having a factor of the form $z = \diamond a^j \diamond$, where $j \geq 1$, satisfies $|w| = 2j + 1 = 2N - 9$.

Proof. We first show that any minimal one-periodic Sturmian partial word w with two holes of order N having a factor of the form $z = \diamond a^j \diamond$, where $j \geq 1$, satisfies $N \geq j + 2$. Suppose not, that is $N < j+2$, so that $N-2 < j$. Set $j = N-2+m$ for some $m \geq 1$. It is easy to note that w avoids $bb, bab, \ldots, ba^{N-2}b$. Setting $\mathcal{S} = \{a^N, a^{N-1}b, a^{N-2}ba, \ldots, aba^{N-2}, ba^{N-1}\}$, we have that $\mathrm{Sub}_z(N) \subset \mathcal{S}$. If w is Sturmian of order N, then $\mathrm{Sub}_w(N) = \mathcal{S}$. Since w is one-periodic, $w = a^i \diamond a^j \diamond a^l$ for some $i, l \geq 0$. If $i + l < N - 2$, then $\mathrm{Sub}_w(N) \neq \mathcal{S}$, so $i + l \geq N - 2$. However, this implies that $|w| \geq N - 2 + N - 2 + m + 2 \geq 2N - 1$. Thus, by Remark 3, w is not minimal, a contradiction.

Now, note that $\mathrm{Sub}_z(j + 2) = \{a^{j+2}, a^{j+1}b, ba^{j+1}, ba^j b\}$ so $p_z(j + 2) = 4$. Suppose $|w| \geq 2j + 2$. Then w has a factor $v = a^i \diamond a^j \diamond a^{j-i}$, for some i, j, $0 \leq i \leq j$. However, we have

$$\mathrm{Sub}_v(j + 2) = \mathrm{Sub}_z(j + 2) \cup \{a^i ba^{j-i+1}, \ldots, aba^j, a^j ba, \ldots, a^{i+1} ba^{j-i}\}$$

Thus, $p_w(j+2) \geq p_v(j+2) = j+4$, a contradiction. Suppose $|w| \leq 2j$. Then $w = a^i \diamond a^j \diamond a^{m-i}$ for some i, j, m, $0 \leq i \leq m < j - 1$. Thus,

$$\mathrm{Sub}_w(j + 2) = \{a^{j+2}, a^{j+1}b, ba^{j+1}, ba^j b, a^i ba^{j-i+1}, \ldots, aba^j, a^j ba, \ldots, a^{j+i-m+1} ba^{m-i}\}$$

so $p_w(j + 2) < 4 + j - 1 = j + 3$, a contradiction. Therefore, $|w| = 2j + 1$.

Note also that $p_w(j+6) \leq 1+\|\mathcal{F}_1(w, j+6)\|+3\|\mathcal{F}_2(w, j+6)\| \leq 1+(|w|-(j+6)-5)+3\times 5 = j + 6 < j + 7$ (there is 1 subword with no b, at most $\|\mathcal{F}_1(w, j + 6)\|$ subwords with one b (fill the hole with b), and at most $3\|\mathcal{F}_2(w, j + 6)\|$ other subwords (fill the holes with ab, ba, bb)). Thus, $j + 2 \leq N < j + 6$. So $N - 5 \leq j \leq N - 2$. The only option is $j = N - 5$ in order to achieve $|w| = 2j + 1 \leq 2N - 9$. Finally, $w = a^{\lfloor (j-1)/2 \rfloor} \diamond a^j \diamond a^{\lceil (j-1)/2 \rceil}$ is of length $2j + 1$ and is Sturmian of order $N = j + 5$ when $j \geq 7$. ◀

Our two-hole description of minimal Sturmian partial words follows.

▶ **Theorem 3.10.** *The only minimal Sturmian partial words with two holes of order $N \geq 12$ are those of the form $a^i \diamond a^j \diamond a^l$, where $j = N - 5$, $i, l \geq 3$, and $i + l = N - 6$, and so $N_2(n + 1, N, 2) = 2N - 22$.*

Proof. Let w be a minimal Sturmian partial word of order N with two holes. The fact that $j = N-5$ and $i+l = N-6$ is evident from Proposition 4. We are left to show that $i, l \geq 3$. Since a^N is trivially a subword of length N of w, we have that $p_w(N) \leq 1 + \|\mathcal{F}_1(w, N)\| + 3\|\mathcal{F}_2(w, N)\|$. Note that since $|w| = 2N - 9$, we have that $\|\mathcal{F}_1(w, N)\| + \|\mathcal{F}_2(w, N)\| \leq |w| - N + 1 = N - 8$. Thus, $\|\mathcal{F}_1(w, N)\| \leq N-8-\|\mathcal{F}_2(w, N)\|$. Therefore, $p_z(N) = N+1 \leq 1+N-8+2\|\mathcal{F}_2(w, N)\|$, implying that $\|\mathcal{F}_2(w, N)\| \geq 4$. Note that if $i < 3$ (the case where $l < 3$ is similar), there are $i + 1 < 4$ factors containing two holes, a contradiction. Thus, there are $N - 11$ hole placements that are valid for a minimal Sturmian partial word of order N with two holes. Since the partial word is one-periodic, we have $N_2(n + 1, N, 2) = 2(N - 11) = 2N - 22$ as desired. ◀

3.4 The case of $h \geq 3$

Recall from Table 1 that $w = \diamond(a^{N-1}\diamond)^{h-1}$ is a Sturmian partial word with h holes of order N of length $N(h+1)+1$, when $h \geq 3$ and $N \geq h+1$. By Remark 3, $L_2(n+1, N, h) \leq N(h-1)+1$ in that case. We show that w is in fact minimal, and that (up to complement) it is the unique such word.

▶ **Lemma 3.11.** *Any Sturmian partial word w having a factor $z = \diamond a^i \diamond a^j \diamond, ba^i \diamond a^j \diamond$, or $\diamond a^i \diamond a^j b$ is of order $N < \min(i,j)+2$. Furthermore, if w has another factor u compatible with $ba^l b$ where $l < \min(i,j)$ then $N < l+2$.*

Proof. Set $t = \min(i,j)$. Assume that $N \geq t+2$. We immediately note that $\mathrm{Sub}_w(t+2) \supset \mathrm{Sub}_z(t+2) = \{a^{t+2}, ba^t b, a^{t+1}b, \ldots, ba^{t+1}\}$, so $\|\mathrm{Sub}_w(t+2)\|$ is at least $t+4$, contradicting the fact that w is Sturmian. Now assume such a factor u exists. Assume that $N \geq l+2$. Trivially, $ba^l b \in \mathrm{Sub}_w(l+2)$. Furthermore, $\{a^{l+2}, a^{l+1}b, \ldots, ba^{l+1}\} \subset \mathrm{Sub}_z(l+2)$. Thus, $p_w(l+2)$ is at least $l+4$, a contradiction. ◀

▶ **Theorem 3.12.** *For $h \geq 3$ and $N \geq h+1$, $L_2(n+1, N, h) = N(h-1)+1$. Furthermore, any minimal Sturmian partial word with h holes of order N is of the form $\diamond(a^{N-1}\diamond)^{h-1}$, and so $N_2(n+1, N, h) = 2$.*

Proof. Any minimal Sturmian partial word w with $h \geq 3$ holes of order N must have a factor of the form $\diamond a^i \diamond$, where $i \geq 1$. By Lemma 3.11, w must be of form $a^{n_0}c_0 a^{n_1}c_1 \cdots a^{n_j}c_j a^{n_{j+1}}$, where each $c_i \in \{\diamond, b\}$ and each $n_i \geq N-1$. It is thus evident that $w = \diamond(a^{N-1}\diamond)^{h-1}$, which was shown in Table 1 to be Sturmian of order N for $N \geq h+1$, is the only form possible for a minimal Sturmian partial word with h holes. ◀

4 Conclusion

We have thus classified all the $(n+1)$-complex partial words with any number of holes. The number of minimal Sturmian full words of order N, $N_2(n+1, N, 0)$, remains to be computed, but an algorithm has been presented that can generate all such words. It would be interesting to complete the classification of the minimal $2n$-complex partial words as well. In this section, we give some preliminary results by filling out Table 2.

Table 2 $2n$-complex partial words with h holes of order N

h	N	partial word	$L_2(2n, N, h)$
0	≥ 3	$a^N ba^{N-2} bba^{N-2}$	$3N-1$
1	≥ 3	$a^{N-2} b\diamond a^{N-2} b$	$2N-1$
2	≥ 5	$a^{\lfloor (N-4)/2 \rfloor} b(\diamond a^{\lceil (N-4)/2 \rceil} b)^2$	$\frac{3N}{2} - 1$ if N is even $\frac{3N}{2} - \frac{1}{2}$ if N is odd
≥ 3	≥ 5	$a^{\lfloor (N-4)/2 \rfloor} b(\diamond a^{\lceil (N-4)/2 \rceil} b)^h$	

▶ Proposition 5. For $h \geq 0$, $f(n) = 2n$ is $(2,h)$-feasible. For $N \geq 3$, $L_2(2n, N, 0) = 3N-1$ and $L_2(2n, N, 1) = 2N-1$.

Follows is our h-hole bound.

▶ Proposition 6. Let w be a word with $h \geq 2$ holes, and $n \leq |w|$ be a positive integer.

- If $|w| \geq 2n-2+h$, then $p_w(n) \leq 2^{h+1} + (n-h+1)2^h + |w| - 2n - h - 2$.

- If $|w| \leq 2n - h$, then $p_w(n) \leq 2^h(|w| - n + 1)$.
- Else $2n - h < |w| < 2n - 2 + h$, and set $d = 2n - 2 + h - |w| > 0$. If d is even, then
$$p_w(n) \leq 2^{h+1} + (n - h + 1)2^h - 4 - 2\sum_{i=1}^{d/2} 2^i = 2^{h+1} + (n - h + 1)2^h - 4 \times 2^{d/2}$$
If d is odd, then
$$p_w(n) \leq 2^{h+1}+(n-h+1)2^h-4-2\sum_{i=1}^{(d-1)/2} 2^i-2^{(d+1)/2} = 2^{h+1}+(n-h+1)2^h-3\times 2^{(d+1)/2}$$

▶ **Corollary 4.1.** *For $N \geq 5$, $L_2(2n, N, 2) = \frac{3N}{2} - 1$ if N is even, and $\frac{3N}{2} - \frac{1}{2}$ if N is odd.*

References

1 M. A. Alekseyev and P. A. Pevzner. Colored de Bruijn graphs and the genome halving problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(1):98–107, 2007.

2 J.-P. Allouche. Sur la complexité des suites infinies. *Bulletin of the Belgium Mathematical Society*, 1:133–143, 1994.

3 J.-P. Allouche and J. Shallit. *Automatic Sequences: Theory, Applications, Generalizations.* Cambridge University Press, 2003.

4 P. Arnoux and G. Rauzy. Représentation géométrique de suites de complexité $2n + 1$. *Bulletin de la Société Mathématique de France*, 119:199–215, 1991.

5 J. Berstel and L. Boasson. Partial words and a theorem of Fine and Wilf. *Theoretical Computer Science*, 218:135–141, 1999.

6 F. Blanchet-Sadri. *Algorithmic Combinatorics on Partial Words.* Chapman & Hall/CRC Press, Boca Raton, FL, 2008.

7 F. Blanchet-Sadri, J. Schwartz, S. Stich, and B. J. Wyatt. Binary de Bruijn partial words with one hole. In J. Kratochvil et al., editor, *TAMC 2010, 7th Annual Conference on Theory and Applications of Models of Computation, Prague, Czech Republic*, volume 6108 of *Lecture Notes in Computer Science*, pages 128–138, Berlin, Heidelberg, 2010. Springer-Verlag.

8 N. G. De Bruijn. Acknowledgement of priority to C. Flye Sainte-Marie on the counting of circular arrangements of $2n$ zeros and ones that show each n-letter word exactly once. Technical Report 75–WSK–06, Department of Mathematics and Computing Science, Eindhoven University of Technology, The Netherlands, 1975.

9 S. Ferenczi. Complexity of sequences and dynamical systems. *Discrete Mathematics*, 206:145–154, 1999.

10 J. L. Gross and J. Yellen. *Handbook of Graph Theory.* CRC Press, 2004.

11 M. Lothaire. *Algebraic Combinatorics on Words.* Cambridge University Press, Cambridge, 2002.

12 F. Manea and C. Tiseanu. Hard counting problems for partial words. In A.-H. Dediu, H. Fernau, and C. Martín-Vide, editors, *LATA 2010, 4th International Conference on Language and Automata Theory and Applications, Trier, Germany*, volume 6031 of *Lecture Notes in Computer Science*, pages 426–438, Berlin, Heidelberg, 2010. Springer-Verlag.

13 M. Morse and G. A Hedlund. Symbolic dynamics II. Sturmian trajectories. *American Journal of Mathematics*, 62(1):1–42, 1940.

14 G. Rote. Sequences with subword complexity $2n$. *Journal of Number Theory*, 46:196–213, 1993.

15 R. P. Stanley. *Enumerative Combinatorics*, volume 2. Cambridge University Press, Cambridge, 2001.

16 M.-w. Wang and J. Shallit. On minimal words with given subword complexity. *The Electronic Journal of Combinatorics*, 5:1–16, 1998.

Improving PPSZ for 3-SAT using Critical Variables

Timon Hertli[1,2], Robin A. Moser[1,3], and Dominik Scheder[1,4]

1 Institute for Theoretical Computer Science
Department of Computer Science
ETH Zürich, 8092 Zürich, Switzerland
2 timon.hertli@inf.ethz.ch
3 robin.moser@inf.ethz.ch
4 dominik.scheder@inf.ethz.ch

Abstract

A critical variable of a satisfiable CNF formula is a variable that has the same value in all satisfying assignments. Using a simple case distinction on the fraction of critical variables of a CNF formula, we improve the running time for 3-SAT from $\mathcal{O}(1.32216^n)$ by Rolf [10] to $\mathcal{O}(1.32153^n)$. Using a different approach, Iwama et al. [5] very recently achieved a running time of $\mathcal{O}(1.32113^n)$. Our method nicely combines with theirs, yielding the currently fastest known algorithm with running time $\mathcal{O}(1.32065^n)$. We also improve the bound for 4-SAT from $\mathcal{O}(1.47390^n)$ [6] to $\mathcal{O}(1.46928^n)$, where $\mathcal{O}(1.46981^n)$ can be obtained using the methods of [6] and [10].

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems; G.2.1 Combinatorics.

Keywords and phrases SAT, satisfiability, randomized, exponential time, algorithm, 3-SAT, 4-SAT

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.237

1 Introduction

The ideas behind the most successful algorithms for k-SAT are surprisingly simple. In 1999, Paturi, Pudlák, and Zane [9] proposed the following algorithm. Given a k-CNF formula F, we choose a variable x uniformly at random from the n variables in F, choose a truth value $b \in \{0,1\}$, and set x to b, thereby replacing F by $F^{[x \mapsto b]}$, and continue with $F^{[x \mapsto b]}$. The value b is chosen as follows: If the formula contains the unit clause (x), we choose $b = 1$. If it contains $(\bar{x})$, we choose $b = 0$. In these two cases, we say x was *forced.* If it contains neither, we choose b randomly and say x was *guessed.* Finally, if the formula contains both (x) and $(\bar{x})$, we can give up, since the formula is unsatisfiable. This algorithm is usually called PPZ after its three inventors.

Intuitively, if F is "strongly constrained", then the algorithm encounters many unit clauses, hence it needs to guess significantly fewer than n variables. On the other hand, if F is only "weakly constrained", it has multiple satisfying assignments, making it easier to find one. Paturi, Pudlák and Zane [9] make this intuition precise and show that PPZ finds a satisfying assignment for a k-CNF formula with probability at least $2^{-(1-1/k)n}$, provided there exists one.

A couple of years later, Paturi, Pudlák, Saks, and Zane [8] came up with a simple but powerful idea. In a preprocessing step, they apply a restricted version of resolution. This increases the number of unit clauses the algorithm encounters and therefore increases its success probability. This gives an algorithm called PPSZ. If F has a unique satisfying assignment, its success probability is quite good (for 3-SAT, it is $\Omega(1.308^{-n})$), and the

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 237–248
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

analysis is highly elegant. The case of multiple satisfying assignments appears to be much more difficult and has been the subject of several papers so far. Iwama and Tamaki [6] made a major step forward when they observed that while the success probability of PPSZ deteriorates as the number of satisfying assignments increases, that of Schöning's random walk algorithm [11] improves. They quantified this tradeoff and obtained an algorithm with a success probability of $\Omega(1.32373^{-n})$[1]. We denote this combined algorithm, consisting of one run of PPSZ and one run of Schöning's random walk algorithm, by Comb.

The PPSZ paper. There are two versions of [8], which we call the old version and the new version. For unique k-SAT, both are the same, but for general k-SAT, the old version of [8] gives a more complicated analysis. The old version gives a better bound for 3-SAT and the new version gives a better bound for 4-SAT.

Only the new version is published, but the old version is still available at the Citeseer cache[2]. However, we have found some minor errors in that version. There is also a conference version [7] stating the results of the old version of [8], but without most proofs. Rolf [10] improved the analysis of the old version to get a bound of $\Omega(1.32216^n)$. However [10] does not consider 4-SAT. We use the ideas of [10] for our improvement of 4-SAT. In Timon Hertli's master thesis [2], the old version of [8] with the result of [10] is presented in a self-contained way. We will reference that thesis for detailed proofs.

1.1 Our Contribution

Let F be a satisfiable CNF formula over n variables and x be a variable therein. We call x *critical* if all satisfying assignments of F agree on x. Equivalently, x is critical if exactly one of the formulas $F^{[x\mapsto 1]}$ and $F^{[x\mapsto 0]}$ is satisfiable. We denote by $c(F)$ the fraction of critical variables, i.e., the number of critical variables divided by n; if $n = 0$, we define $c(F) := 1$.

Our contribution consists of two statements: Theorem 1 shows that for our purposes we only need to consider formulas with many critical variables. Point 3 of Lemma 9 then implies that the success probability of PPSZ increases if F has many critical variables. This is obtained by slightly modifying the existing analysis of [8] and [10] by taking critical variables into account. However, Lemma 9 is somewhat technical and we need to embed it into a review of the existing analysis. Theorem 1 is very simple, so we state it here:

▶ **Theorem 1.** *Let $p, q, c^* \in [0,1]$ and $a, b \geq 1$ such that $\frac{q}{b} = \left(1 - \frac{c^*}{2}\right) =: r$. Suppose algorithm $\mathcal{A}$ runs in time $a^n 2^{o(n)}$ and for every satisfiable $(\leq k)$-CNF formula F with $c(F) \geq c^*$ finds a satisfying assignment with probability at least $p^n \left(\frac{1}{2}\right)^{o(n)}$. Then there exists an algrotihm $\mathcal{A}'$ that runs in time $\max\{a,b\}^n 2^{o(n)}$ and for every satisfiable $(\leq k)$-CNF formula finds a satisfying assignment with probability at least $\min\{p,q\}^n \left(\frac{1}{2}\right)^{o(n)}$.*

Obviously we can turn $\mathcal{A}'$ into a algorithm that finds a satisfying assignment in expected time $\left(\frac{\max\{a,b\}}{\min\{p,q\}}\right)^n 2^{o(n)}$.

Proof. By *guessing* j variables we mean fixing in F j variables chosen uniformly at random to values chosen uniformly at random, obtaining the formula F' over at most $n - j$ variables. $\mathcal{A}'$ for each $j \in \{0, \ldots, n\}$ repeats the following b^j times: Guess j variables and then run $\mathcal{A}$ on F'; the running time bound is trivial. To bound the probability, we first claim that there

[1] Using the new version of [8] immediately gives the bound $\Omega(1.32267^{-n})$, as stated in [10].
[2] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.1134

exists a j such that $a_j \geq \frac{r^j}{n+1}$ where a_j is the probability that after guessing j variables F' is satisfiable and $c(F') \geq c^*$. Suppose this is not the case: Let b_j be the probability that after guessing j variables F' is satisfiable and $c(F') < c^*$. Clearly $a_0 + b_0 = 1$ since F is satisfiable, and $a_{i+1} + b_{i+1} \geq b_i \cdot r$, as guessing one variable preserves satisfiability with probability at least $\left(1 - \frac{c^*}{2}\right) = r$. By the assumption, $b_i \cdot r \geq \left(a_i + b_i - \frac{r^i}{n+1}\right) \cdot r$; from this it is easy to show that $a_n + b_n \geq r^n - n\frac{r^n}{n+1} = \frac{r^n}{n+1}$. If $j = n$, we have $c(F') = 1$ by definition; hence $b_n = 0$ and $a_n \geq \frac{r^n}{n+1}$, a contradiction. Now let j^* be the j given by the claim; we repeat b^{j^*} times an algorithm that has success probability at least $\frac{r^{j^*}}{n+1} p^{n-j^*} \left(\frac{1}{2}\right)^{o(n)}$; as $r \cdot b = q$ this gives by a routine argument an algorithm with success probability at least $p^{n-j^*} q^{j^*} \left(\frac{1}{2}\right)^{o(n)}$. ◀

We improve the analysis for PPSZ for formulas with many critical variables. In combination with Theorem 1, this gives a success probability of $\Omega(1.32153^{-n})$ for 3-SAT and $\Omega(1.46928^{-n})$ for 4-SAT. Very recently, Iwama, Seto, Takai, and Tamaki [5] showed how to combine an improved version of Schöning's algorithm [4, 1] with PPSZ and achieved expected running time of $O(1.32113^n)$. We combine our improvement with theirs to obtain a bound of $O(1.32065^n)$. Due to page limitations, we were not able to use the full power of [5] in this version. We show a bound $O(1.321^n)$ that still improves on the bound of [5]. For a proof of the better bound, see the full version of this paper [3]. The only change is we use a better result of [5] which has different parameters; however these are not not stated explicitly so we needed to derive and prove them.

We analyze the algorithm Comb(F), where F is a CNF formula. Comb consists essentially of a call to PPSZ [8] and to Schoening [11]. In [6] it was shown that Comb has a better success probability than what the analysis of PPSZ and Schoening gives. Let ISTT be the algorithm of [5] that improves Comb.

▶ **Theorem 2.** *There exists an algorithm that for every satisfiable* 3*-CNF formula finds a satisfying assignment with probability* $\Omega(1.32153^{-n})$ *and runs in subexponential time.*

▶ **Theorem 3.** *There exists an algorithm that for every satisfiable* 3*-CNF formula finds a satisfying assignment with expected running time* $O(1.32065^{-n})$.

Due to page limitations, we prove the following weaker theorem instead. For the proof of the previous theorem, see the full version of this paper [3].

▶ **Theorem 4.** *There exists an algorithm that for every satisfiable* 3*-CNF formula finds a satisfying assignment with expected running time* $O(1.321^{-n})$.

▶ **Theorem 5.** *There exists an algorithm that for every satisfiable* 3*-CNF formula finds a satisfying assignment with probability* $\Omega(1.46928^{-n})$ *and runs in subexponential time.*

This is already very close to unique 4-SAT, which has a success probability of $\Omega(1.46899^{-n})$. The benefit of Theorem 1 is that when proving Theorems 2 and 5, we only need to consider formulas with many critical variables. For example, to prove Theorem 2, we choose c^* such that $1 - c^*/2 = 1/1.32153$, i.e., $c^* \approx 0.4866$. Then we have to bound from below the success probability of Comb for 3-CNF formulas F with $c(F) \geq c^*$.

1.2 Notation

We use the notational framework introduced in [12]. We assume an infinite supply of propositional *variables*. A *literal* u is a variable x or a complemented variable $\bar{x}$. A finite set

C of literals over pairwise distinct variables is called a *clause* and a finite set of clauses is a formula in *CNF* (Conjunctive Normal Form). We say that a variable x *occurs* in a clause C if either x or $\bar{x}$ are contained in it and that x occurs in the formula F if there is any clause where it occurs. We write $\mathrm{vbl}(C)$ or $\mathrm{vbl}(F)$ to denote the set of variables that occur in C or in F, respectively. A clause containing exactly one literal is called a *unit clause*. We say that F is a *$(\leq k)$-CNF* formula if every clause has size at most k. Let such an F be given and write $V := \mathrm{vbl}(F)$ and $n := |V|$.

A *assignment* is a function $\alpha : V \to \{0, 1\}$ which assigns a Boolean value to each variable. A literal $u = x$ (or $u = \bar{x}$) is *satisfied by* α if $\alpha(x) = 1$ (or $\alpha(x) = 0$). A clause is *satisfied by* α if it contains a satisfied literal and a formula is *satisfied by* α if all of its clauses are. A formula is *satisfiable* if there exists a satisfying truth assignment to its variables.

For an assignment α on V and a set $W \subseteq V$, we denote by $\alpha \oplus W$ the assignment that corresponds to α on variables of $V \setminus W$ and is flipped on variables of W.

Given a CNF formula F, we denote by $\mathrm{sat}(F)$ the set of assignments that satisfy F.

Formulas can be manipulated by permanently assigning values to variables. If F is a given CNF formula and $x \in \mathrm{vbl}(F)$ then assigning $x \mapsto 1$ satisfies all clauses containing x (irrespective of what values the other variables in those closes are possibly assigned later) whilst it truncates all clauses containing $\bar{x}$ to their remaining literals.

We will write $F^{[x\mapsto 1]}$ (and analogously $F^{[x\mapsto 0]}$) to denote the formula arising from doing just this.

We say that two clauses C_1 and C_2 conflict on a variable x if one of them contains x and the other $\overline{x}$. We call C_1 and C_2 a resolvable pair if they conflict in exactly one variable x, and we define their *resolvent* by $R(C_1, C_2) := (C_1 \cup C_2) \setminus \{x, \overline{x}\}$. It is easy to see that if F contains a resolvable pair C_1, C_2, then $\mathrm{sat}(F) = \mathrm{sat}(F \cup \{R(C_1, C_2)\})$. A resolvable pair C_1, C_2 is s-bounded if $|C_1| \leq s$, $|C_2| \leq s$, and $|R(C_1, C_2)| \leq s$.

By $\textsc{Resolve}(F, s)$, we denote the set of clauses C that have an s-bounded resolution deduction from F. By a straightforward algorithm, we can compute $\textsc{Resolve}(F, s)$ in time $O\left(n^{3s}\mathrm{poly}\left(n\right)\right)$ [8].

By choosing an element u.a.r. from a finite set, we mean choosing it uniformly at random. By choosing an element u.a.r. from an closed real interval, we mean choosing it according to the continuous uniform distribution over this interval. Unless otherwise stated, all random choices are mutually independent.

We denote by log the logarithm to the base 2. For the logarithm to the base e, we write ln. We define $0 \log 0 := 0$.

2 Proof of the Main Theorems

In the following let $k \geq 3$ be a fixed integer. Let F be a satisfiable $(\leq k)$-CNF formula, $V := \mathrm{vbl}(F)$ and $n := |V|$. We first give the concepts from [8] needed to understand Lemma 9. Then we state the lemma and use it to improve the bounds on the success probability of $\textsc{Comb}$ and ISTT given sufficiently many critical variables. In Section 3, we prove Lemma 9 and also consider 4-SAT. Most concepts used in the proof are from [8, 10]. Our contribution is to exploit what these concepts yield for critical variables.

Subcubes. For $D \subseteq V$ and $\alpha \in \{0, 1\}^V$, the set $B(D, \alpha) := \{\beta \in \{0, 1\}^V \mid \alpha(x) = \beta(x)\ \forall x \in D\}$ is called a *subcube*. The variables in D are called *defining* variables and those in $V \setminus D$ *nondefining* variables. The subcube $B(D, \beta)$ has dimension $|V \setminus D|$. For example, if $V = \{x_1, x_2, x_3\}$, $D = \{x_1, x_3\}$ and $\alpha = (1, 0, 0)$, then $B(D, \alpha)$ contains exactly the two

Algorithm 1 PPSZ(CNF formula F, assignment β, permutation π)

```
Let $\alpha$ be a partial assignment over vbl($F$), initially the empty assignment.
$G \leftarrow$ Resolve($F, \log(|\mathrm{vbl}(F)|)$)
for all $x \in \mathrm{vbl}(G)$, according to $\pi$ do
  if $\{x\} \in G$ then
    $\alpha(x) \leftarrow 1$
  else if $\{\bar{x}\} \in G$ then
    $\alpha(x) \leftarrow 0$
  else
    $\alpha(x) \leftarrow \beta(x)$
  end if
  $G \leftarrow G^{[x \mapsto \alpha(x)]}$
end for
return $\alpha$
```

Algorithm 2 PPSZ(CNF formula F)

```
{this algorithm is used for 4-SAT}
Choose $\beta(x)$ u.a.r. from all assignments on vbl($F$)
Choose $\pi$ u.a.r. from all permutations of vbl($F$)
return PPSZ($F, \beta, \pi$)
```

assignments $(1, 0, 0)$ and $(1, 1, 0)$. Given a nonempty set $S \subseteq \{0, 1\}^V$, there is a partition

$$\{0,1\}^V = \bigcup_{\alpha \in S} B_\alpha$$

where the B_α are pairwise disjoint subcubes, and $\alpha \in B_\alpha$ for all $\alpha \in S$. See [8] for a proof. For the rest of the paper, we fix such a partition for S being the set of satisfying assignments. To estimate the success probability of Comb, consider the assignment β that Comb chooses uniformly at random from $\{0, 1\}^V$.

$$\begin{aligned}\Pr[\textsc{Comb}(F) \in \mathrm{sat}(F)] &= \sum_{\alpha \in \mathrm{sat}(F)} \Pr[\textsc{Comb}(F) \in \mathrm{sat}(F)|\ \beta \in B_\alpha] \cdot \Pr[\beta \in B_\alpha] \\ &\geq \min_{\alpha \in \mathrm{sat}(F)} \Pr[\textsc{Comb}(F) \in \mathrm{sat}(F) \mid \beta \in B_\alpha].\end{aligned}$$

Hence instead of analyzing Comb for an assignment β sampled uniformly at random from all assignments, we fix $\alpha \in \mathrm{sat}(F)$ arbitrarily and we think of β as being sampled from the subcube B_α. Let N_α be the set of non-defining variables of this cube, and D_α the set of defining variables. Intuitively, if B_α has small dimension, then β is likely to be close to α, thus Schoening has a better success probability:

▶ **Lemma 6** ([6]). $\Pr[\textsc{Schoening}(F, \beta) \in \mathrm{sat}(F) \mid \beta \in B_\alpha] \geq (2 - 2/k)^{-|N_\alpha|}$.

Placements. As a next step, we analyze PPSZ(F, β, π) with β chosen uniformly at random from B_α and the permutation also chosen from some subset of permutations. A *placement* of the variables V is a function $\sigma : V \to [0, 1]$, and a *uniform random placement* is defined by chosing $\sigma(x)$ uniformly at random from $[0, 1]$ independently for each $x \in V$. With probability 1, a uniform random placement is injective and gives rise to a uniformly distributed permutation via the natural ordering $<$ on $[0, 1]$. For the rest of the paper, we will

Algorithm 3 SCHOENING(CNF formula F, assignment β)

```
for $3|\text{vbl}(F)|$ steps do
  if $\beta$ satisfies $F$ then
    return $\beta$
  end if
  Select an arbitrary $C \in F$ not satisfied by $\beta$
  Select a variable $x$ u.a.r. from $\text{vbl}(C)$ and flip $x$ in $\beta$
end for
return $\beta$
```

Algorithm 4 COMB(CNF formula F)

```
{this algorithm is used for 3-SAT}
Choose $\beta(x)$ u.a.r. from all assignments on $\text{vbl}(F)$
$\alpha \leftarrow \text{PPSZ}(F, \beta)$
if $\alpha \notin \text{sat}(F)$ then
  $\alpha \leftarrow \text{SCHOENING}(F, \beta)$
end if
return $\alpha$
```

view π as a placement rather than a permutation. Let Γ be a measurable set of placements. Then

$$\Pr[\text{PPSZ}(F, \beta, \pi) \in \text{sat}(F) \mid \beta \in B_\alpha] \geq \\ \Pr[\text{PPSZ}(F, \beta, \pi) \in \text{sat}(F) \mid \beta \in B_\alpha, \pi \in \Gamma] \cdot \Pr[\pi \in \Gamma].$$

The benefit of this is that we can tailor Γ towards our needs, i.e., making the conditional probability $\Pr[\text{PPSZ}(F, \beta, \pi) \in \text{sat}(F) \mid \beta \in B_\alpha, \pi \in \Gamma]$ fairly large. This may come at the cost of making $\Pr[\pi \in \Gamma]$ small.

Forced variables. Suppose the permutation π orders the variables V as $(x_1, \ldots, x_n)$. Let α be a satisfying assignment of F. Imagine we call $\text{PPSZ}(F, \alpha, \pi)$. The algorithm applies bounded resolution to F, obtaining $G = \text{RESOLVE}(F, \log(n))$ and sets the variables $x_1, \ldots, x_n$ step by step to their respective values under α, creating a sequence of formulas by $G = G_0, G_1, \ldots, G_n$, where $G_i = G_{i-1}^{[x_i \mapsto \alpha(x_i)]}$ for $1 \leq i \leq n$. Since α is a satisfying assignment, G_n is the empty formula. We say x_i is *forced* with respect to α and π if G_{i-1} contains the unit clause $\{x_i\}$ or $\{\bar{x}_i\}$. By $\text{forced}(\alpha, \pi)$ we denote the set of variables x that are forced with respect to α and π. If x is not forced, we say it is *guessed.* We denote by $\text{guessed}(\alpha, \pi)$ the set of guessed variables. Note that $\text{PPSZ}(F, \beta, \pi)$ returns α if and only if $\alpha(x) = \beta(x)$ for all $x \in \text{guessed}(\alpha, \pi)$. Furthermore, since β is chosen uniformly at random from B_α, we already have $\alpha(x) = \beta(x)$ for all $x \in D_\alpha$. Therefore

$$\begin{aligned} \Pr[\text{PPSZ}(F, \beta, \pi) \in \text{sat}(F)] &\geq \Pr[\text{PPSZ}(F, \beta, \pi) = \alpha] & (1) \\ &= \mathbf{E}\left[2^{-|N_\alpha \cap \text{guessed}(\alpha, \pi)|}\right] \\ &\geq 2^{-\mathbf{E}[|N_\alpha \cap \text{guessed}(\alpha, \pi)|]}, & (2) \end{aligned}$$

where the inequality comes from Jensen's inequality applied to the convex function $t \mapsto 2^{-t}$. Note that (2) holds when taking π uniformly at random as well as when sampling it from some set Γ. Using linearity of expectation, we see that

$$\mathbf{E}[|N_\alpha \cap \text{guessed}(\alpha,\pi)|] = \sum_{x \in N_\alpha} \Pr[x \in \text{guessed}(\alpha,\pi)]. \tag{3}$$

Now if α is the unique satisfying assignment, then $N_\alpha = V$. For 3-SAT, one central result of [8] is that

▶ **Lemma 7** ([8]). *Let F be a satisfiable 3-CNF formula with a unique satisfying assignment α. Then for every $x \in \text{vbl}(F)$, it holds that $\Pr[x \in \text{guessed}(\alpha,\pi)] \leq 2\ln(2) - 1 + o(1) < 0.3863$.*

Combining the lemma with (2) shows that PPSZ on 3-CNF formulas with a unique satisfying assignment has a success probability of at least $2^{-(2\ln(2)-1+o(1))n} \in \Omega(1.308^{-n})$. For the case of multiple satisfying assignments, the lemma does not hold anymore.

Critical variables. Let F be a satisfiable CNF formula and x a variable. Recall that we call x *critical* if all satisfying assignments of F agree on x. The following observation is not difficult to show:

▶ **Observation 8.** *Let F be a satisfiable CNF formula and let V_C be the set of critical variables. Let B_α be the subcube as defined above. For a satisfying assignment α, let N_α be the set of nondefining variables. Then $V_C \subseteq N_\alpha$.*

▶ **Lemma 9.** *Let F be a satisfiable 3-CNF formula and α be a satisfying assignment. There is a measurable set $\Gamma \subseteq [0,1]^V$ of placements such that for $\beta = 0.8022563838$ and $\gamma = 0.6073995502$, we have*

1. $\Pr[\pi \in \Gamma] \geq 2^{-\beta|D_\alpha| - o(n)} \approx 0.57345159^{|D_\alpha| - o(n)}$,
2. $\Pr[x \in \text{forced}(\alpha,\pi) \mid \pi \in \Gamma] \geq \gamma - o(1) \approx 0.6073995502 - o(1)$ *for all* $x \in N_\alpha$,
3. $\Pr[x \in \text{forced}(\alpha,\pi) \mid \pi \in \Gamma] \geq 2 - 2\ln(2) - o(1) \approx 0.6137056$ *for all critical* $x \in V$.

The important part of the lemma is point 3, namely that critical variables are forced with a larger probability than non-critical ones.

Proof of Theorem 2. Using Theorem 1, we can assume $c(F) \geq 0.48659459$. Let $\Delta := |D_\alpha|/|V| = 1 - |N_\alpha|/|V|$ be the fraction of defining variables. Combining (3) with Lemma 9, we obtain

$$\begin{aligned}
\mathbf{E}[|N_\alpha \cap \text{guessed}(\alpha,\pi)| \mid \pi \in \Gamma] &= \sum_{x \in N_\alpha} \Pr[x \in \text{guessed}(\alpha,\pi)] \\
&\leq (2\ln 2 - 1)|V_C| + (1-\gamma)|N_\alpha \setminus V_C| + o(n) \\
&\leq (2\ln 2 - 1)c^* n + (1-\gamma)(1 - \Delta - c^*)n + o(n) \\
&= 0.389532n - 0.3926004498\Delta n + o(n).
\end{aligned}$$

The expected fraction of nondefining variables we have to guess is thus a little bit larger than in the case of a unique satisfying assignment, where it is ≈ 0.3863. Together with (2), we conclude that the success probability of PPSZ is at least

$$\begin{aligned}
\Pr[\text{PPSZ}(F,\beta,\pi) = \alpha \mid \beta \in B_\alpha] &\geq \Pr[\text{PPSZ}(F,\beta,\pi) = \alpha \mid \beta \in B_\alpha, \pi \in \Gamma] \cdot \Pr[\pi \in \Gamma] \\
&\geq 2^{-\mathbf{E}[|N_\alpha \cap \text{guessed}(\alpha,\pi)| \mid \pi \in \Gamma]} \cdot \Pr[\pi \in \Gamma] \\
&\geq 2^{-0.389532n + 0.3926004498\Delta n} \cdot 0.57345159^{\Delta n} \cdot 2^{-o(n)} \\
&\geq 1.3099684^{-n} \cdot 1.328369^{-\Delta n} \cdot 2^{-o(n)}. \qquad (4)
\end{aligned}$$

Our bound on the success probability of PPSZ thus deteriorates with the number of defining variables. A bigger subcube B_α is better for PPSZ. We combine this with the bound for Schöning's algorithm from Iwama and Tamaki [6], stated above in Lemma 6

$$\Pr[\textsc{Schoening}(F, \beta) \in \mathrm{sat}(F) \mid \beta \in B_\alpha] \geq (2 - 2/k)^{-(1-\Delta)n}. \tag{5}$$

The combined worst case is with $\Delta \approx 0.0309273$, in which case both (4) and (5) evaluate to $\Omega(1.32153^{-n})$. Therefore for any Δ, at least one of Schoening and PPSZ has a success probability of $\Omega(1.32153^{-n})$. ◀

Proof of Theorem 4. Lemma 6 from [5] tells us that there is an algorithm ISTTSch that improves Schoening such that for all $m^* \in [0, \frac{1}{3}]$ we have, after preprocessing time $6^{m^* n}$,

$$\Pr[\text{ISTT}\textsc{Sch}(F, \beta) \in \mathrm{sat}(F) \mid \beta \in B_\alpha] \geq 1.012795^{m^* \cdot n} \cdot 1.2845745^{\Delta n} \cdot (3/4)^n.$$

We want to prove that by replacing Schoening with ISTTSch in Comb, we obtain expected running time of $O(1.321^n)$. Setting $c^* := 0.48599$ and $m^* := 0.155371873$ gives $1 - c^*/2 \geq 1/1.321$ and $6^{m^*} \geq 1.321$. With this choice of c^*, we have the following bound for PPSZ (obtained as in the previous proof, but with a different constant c^*):

$$\Pr[\mathrm{PPSZ}(F, \beta, \pi) = \alpha \mid \beta \in B_\alpha] \geq 1.31^{-n} \cdot 1.3312^{-\Delta n} \cdot 2^{-o(n)}.$$

The combined worst case is at $\Delta \approx 0.029225$ where $1.31^{-n} \cdot 1.3312^{-\Delta n} > 1.321^{-n}$ and $1.012795^{m^* \cdot n} \cdot 1.2845745^{\Delta n} \cdot (3/4)^n > 1.321^{-n}$, proving that the combined success probability is $\Omega(1.321^{-n})$ (after preprocessing time $O(1.321^n)$). ◀

3 Proof of Lemma 9

3.1 Critical Clause Trees

Let $G := \textsc{Resolve}(F, \log(n))$. Note that $\mathrm{vbl}(F) = \mathrm{vbl}(G)$ and $\mathrm{sat}(F) = \mathrm{sat}(G)$. A *critical clause* for $x \in V$ w.r.t. α is a clause where α satisfies exactly one literal and this literal is over x. It can be easily seen that if the output of PPSZ should be α, then exactly the critical clauses of G are the clauses that might turn into unit clauses. Note that the *defining* variables are assumed to be set correctly, so we only need to consider critical clauses for *nondefining* variables here.

We now define critical clause trees, a concept that tells us which critical clauses we can expect in a CNF formula after bounded resolution. Let T be a rooted tree in which every node is either labeled with a variable from V or is unlabeled. A *cut* in a rooted tree is a set of nodes A such that the root is not in A and every path from the root to a leaf contains at least one node in A. The *depth* of a node is the distance to the root. For a set A of nodes, $\mathrm{vbl}(A)$ denotes the set of variables occurring as labels in A. We say T is a *critical clause tree* for x w.r.t. G and α if the following properties hold:

1. The root is labeled by x.
2. On any path from the root to a leaf, no two nodes have the same label.
3. For any cut A of the tree, there is a critical clause $C \in G$ w.r.t. α where the satisfied literal is over x and every unsatisfied literal is over some variable in $\mathrm{vbl}(A)$.

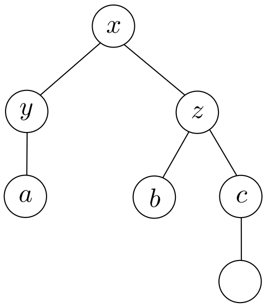


Figure 1 Example Critical Clause Tree

It is shown in [8] that we can construct a critical clause tree for $x \in N_\alpha$ as follows: Start with the root labeled x. Now we can repeatedly extend a leaf node v. Let L be the set of labels that occur on the path from v to the root. If $\alpha \oplus L$ does not satisfy F, then we can extend the tree at that node: There is a clause C in F (not in G) not satisfied by $\alpha \oplus L$. For each literal in C that is not satisfied by α, we add a child to v labeled with the variable of that literal. If there are no such literals, we add an unlabeled node. As clauses of F have at most k literals, each node has at most $k-1$ children. If the constructed tree has at most $\log(n)$ nodes (as we do $\log(n)$-bounded resolution), then it is a critical clause tree for x w.r.t. G and α.

We give a simple example: Let

$$F := \{\{x, \bar{y}, \bar{z}\}, \{x, y, \bar{a}\}, \{z, \bar{b}, \bar{c}\}, \{x, z, c\}\}.$$

For the all-one assignment and x, we can get the tree shown in Figure 1 by the described procedure. $\{a, b\}$ is a cut in this tree. We have $R(\{z, \bar{b}, \bar{c}\}, \{x, z, c\}) = \{x, z, \bar{b}\}$, $R(\{x, \bar{y}, \bar{z}\}, \{x, y, \bar{a}\}) = \{x, \bar{z}, \bar{a}\}$ and $R(\{x, z, \bar{b}\}, \{x, \bar{z}, \bar{a}\}) = \{x, \bar{a}, \bar{b}\}$, giving the required critical clause.

If α is the only satisfying assignment of F, $\alpha \oplus L$ never satisfies F, and we can build a tree where all leafs are at depth $d := \lfloor \log_k \log(n)) \rfloor$. We call this a *full tree*. The important observation is now that this also works if x is a *critical variable*, as in that case $\alpha \oplus L$ also never satisfies F, as $x \in L$.

In the general case, however, the assignment $\alpha \oplus L$ might satisfy F so that we cannot extend the tree. However if L consists only of *nondefining* variables, then we know that $\alpha \oplus L$ does not satisfy F. Hence we can get a tree where every leaf not at depth d is labeled by a *defining* variable. We define the trees T_x we will use in the analysis:

▶ **Definition 10.** For $x \in N_\alpha$, construct the critical clause tree for x as follows: If x is a critical variable, then construct T_x such that all leaves are at depth d, i.e., construct a full tree. Otherwise, construct T_x such that all leaves not labeled by defining variables are at depth d.

This means that a tree might just consist of a root where all children are labeled with defining variables, which essentially nullifies the benefits from resolution. To cope with this, we have to make defining variables more likely to occur at the beginning. We achieve this by choosing the set Γ of placements whose existence we claim in Lemma 9 in a way such that exactly that happens.

▶ **Definition 11.** A function $H : [0, 1] \to [0, 1]$ is called a *nice distribution function* if H is non-decreasing, uniformly continuous, $H(0) = 0$, $H(1) = 1$, H is differentiable except for finitely many points and $H(r) \geq r$.

Compared with [8], we added the requirement $H(r) \geq r$. This will mean that defining variables cannot be less likely to occur at the beginning than nondefining variables. We now define a random placement where defining variables are placed with distribution function H:

▶ **Definition 12.** Let H be a nice distribution function. By π_H, we define the random placement on V s.t. $\pi(x)$ for $x \in N_\alpha$ is u.a.r. $\in [0, 1]$, and for $x \in D_\alpha$ and $r \in [0, 1]$, $\Pr(\pi(x) \leq r) = H(r)$.

Assume that the variables are processed according to some placement π. Consider T_x. If there is a cut A such that $\pi(y) < \pi(x)$ for every $y \in \mathrm{vbl}(A)$, then x is forced, as the corresponding critical clause has turned into a unit clause for x. Denote the probability that $S_x(\pi)$ is a cut in T_x by $Q(T_x, \pi)$.

For $r \in [0,1]$, let $R_k(r)$ be the smallest non-negative x that satisfies $x = (r + (1-r)x)^{k-1}$ and $R_k := \int_0^1 R_k(r)dr$. It was shown in [8] that if T_x is a full tree, then

$$Q(T_x, \pi_U) \geq R_k - o(1).$$

$R_k(r)$ can be understood as follows: Take an infinite $(k-1)$-ary tree and mark each node as "dead" with probability r, except the root. $R_k(r)$ is the probability that this tree contains an infinite path that starts at the root and contains only "alive" nodes.

We have $R_3 = 2 - 2\ln 2 \approx 0.6137$ and $R_4 \approx 0.4451$. For $r \in [0, \frac{1}{2}]$, we have $R_3(r) = \left(\frac{r}{1-r}\right)^2$ and for $r \in [\frac{1}{2}, 1]$, we have $R_3(r) = 1$. As $H(r) \geq r$, and by definition of π_H and of a cut, it is obvious that

$$Q(T_x, \pi_H) \geq R_k - o(1), \tag{6}$$

if T_x is a full tree. If T_x is not a full tree, we do not have any good bounds on $Q(T_x, \pi_U)$. In [10] it is shown that if T_x is not necessarily a full tree, but a tree in which every leaf not at depth d is labeled by a defining variable, then

$$Q(T_x, \pi_H) \geq \gamma_H - o(1), \tag{7}$$

where

$$\gamma_H = \int_0^1 \min\{H(r)^{k-1}, R_k(r)\}dr.$$

Obviously $\gamma_H \leq R_k$, which means that the bound (6) for full trees is at least as strong as the bound (7) for general trees. The $H(r)^{k-1}$ term corresponds to the tree that consists of a root where all children are labeled with defining variables and are thus leaves (remember that there are at most $k-1$ children). It takes a small lemma to show that this tree and the full tree are the worst cases. See [2] for details. The following observation summarizes this:

▶ **Observation 13.** *If x is a* critical *variable, then $Q(T_x, \pi_H) \geq R_k - o(1)$. If x is a noncritical* nondefining *variable, then $Q(T_x, \pi_H) \geq \gamma_H - o(1)$.*

We want to find a set Γ of placements such that a placement chosen uniformly at random from Γ behaves more or less like π_H.

▶ **Lemma 14** (old version of [8]). *Let H be a nice distribution function. If $|D_\alpha| \geq \sqrt{n}$, there is a set of placements Γ depending on n with the following properties: Let π_Γ be the placement choosen uniformly at random from Γ. Then for any tree T with at most $\log(n)$ nodes we have*

$$Q(T, \pi_\Gamma) \geq Q(T, \pi_H) - o(1)$$

and

$$Pr(\pi_U \in \Gamma) \geq 2^{-\beta_H |D_\alpha| - o(n)}$$

with

$$\beta_H := \int_0^1 h(r) \log\left(h(r)\right) dr$$

where $h(r)$ is the derivative of $H(r)$.

The proof of this lemma is long and complicated, see Sections 4.2 and 4.3 in [2]. The case $|D_\alpha| < \sqrt{n}$ is easy to handle: The probability that all defining variables come at the beginning is substantial, and we are essentially in the (good) unique case.

Below we will show how to choose a good function H for the case $k = 3$ and $k = 4$. To get an intuition, see Figure 2 for a plot of H for $k = 3$. With this function, one obtains $\gamma_H \approx 0.6073995502$ and $\beta_H \approx 0.8022563838$. Together with Lemma 14 and Observation 13, we conclude that for a *critical* variable x

$$\Pr[x \in \text{forced}(\alpha, \pi)] \geq Q(T_x, \pi_H) - o(1) \geq R_k - o(1) \geq 0.61371,$$

and for a non-critical non-defining variable x

$$\Pr[x \in \text{forced}(\alpha, \pi)] \geq Q(T_x, \pi_H) \geq \gamma_H - o(1) \geq 0.6073995502 - o(1).$$

3.2 Choosing a good H

3-SAT. Let now $k = 3$. We choose H as in [10]: Let $\theta \in [0.5, 1]$ be a parameter. With some appropriate parameters a and $b > 1$, we define $H(r)$ as follows:

$$H(r) := \begin{cases} r/\theta & \text{if } r \in [0, 1-\theta) \\ 1 - (-a\ln(r))^b & \text{if } r \in [1-\theta, 1] \end{cases}$$

To determine a and b, we set the constraints

$$H(1-\theta) = R_3(1-\theta)^{1/2}$$

(as $\theta \geq 1/2$, this right-hand side is equal to $\frac{1-\theta}{\theta}$) and

$$h(1-\theta) = 1/\theta.$$

If these constraints are satisfied, $H(r)$ is a nice distribution function that is differentiable on $[0, 1]$. Figure 2 gives a plot of the $H(r)$ we use. Numerical optimization gives $\theta \approx 0.52455825$ and as before $c^* \approx 0.48659459$. See Section 4.6 in [2] for details of the computation. This gives

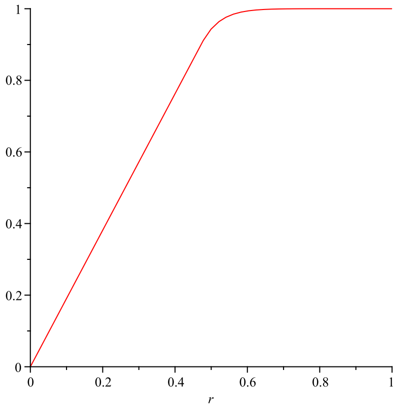


Figure 2 $H(r)$ for 3-SAT

$$a \approx 0.96782885577,$$

$$b \approx 7.19709520894,$$

$$\beta_H \leq 0.8022563838,$$

$$\gamma_H \geq 0.6073995502.$$

This concludes the proof of Lemma 9.

4-SAT. For 4-SAT, we use the H corresponding to the new version of [8]. For some parameter $\theta \in [\frac{2}{3}, 1]$, we let $H(r) := \min\{\frac{r}{\theta}, 1\}$. It turns out that the optimum is when $\beta_H = 1 - \gamma_H$. In that case it is easily seen that the bound for PPSZ does not depend on $|D_\alpha|$, and hence we do not need SCHOENING. Numerical optimization gives $\theta \approx 0.6803639$ and $c^* \approx 0.63878808$. This implies the success probability $\Omega(1.46928^{-n})$, proving Theorem 5.

4 Conclusion

We have shown how to improve PPSZ by a preprocessing step that guarantees that a substantial fraction of variables will be critical. With this, we were able to improve the bound for 3-SAT and 4-SAT from [10]. We have also shown that our approach nicely combines with the improvement by [5] by giving an even better bound. In 4-SAT, we are already very close to the unique case. We do not know if a more refined choice of H (similar to [10]), possibly depending on Δ, allows us to close that gap.

It is interesting to see that we could make use of multiple assignments in the guessing step before considering just one assignment using the subcube partition.

Acknowledgments

We thank Emo Welzl for many fruitful discussions and continous support and Konstantin Kutzkov for pointing us to [5].

References

1 Sven Baumer and Rainer Schuler. Improving a probabilistic 3-SAT algorithm by dynamic search and independent clause pairs. In *Theory and Applications of Satisfiability Testing*, volume 2919 of *Lecture Notes in Computer Science*, pages 150–161. Springer Berlin / Heidelberg, 2004.

2 Timon Hertli. Investigating and improving the PPSZ algorithm for SAT, master's thesis. ETH Zürich, 2010. doi: `http://dx.doi.org/10.3929/ethz-a-006206989`.

3 Timon Hertli, Robin A. Moser, and Dominik Scheder. Improving PPSZ for 3-SAT using critical variables. *CoRR*, abs/1009.4830, 2010.

4 Thomas Hofmeister, Uwe Schöning, Rainer Schuler, and Osamu Watanabe. A probabilistic 3-SAT algorithm further improved. In *STACS 2002*, volume 2285 of *Lecture Notes in Comput. Sci.*, pages 192–202. Springer, Berlin, 2002.

5 Kazuo Iwama, Kazuhisa Seto, Tadashi Takai, and Suguru Tamaki. Improved randomized algorithms for 3-SAT. In *Algorithms and Computation*, volume 6506 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin / Heidelberg, 2010.

6 Kazuo Iwama and Suguru Tamaki. Improved upper bounds for 3-SAT. In *Proceedings of the Fifteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 328–329 (electronic), New York, 2004. ACM.

7 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An Improved Exponential-Time Algorithm for k-SAT. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 628–637. IEEE Computer Society, 1998.

8 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364 (electronic), 2005.

9 Ramamohan Paturi, Pavel Pudlák, and Francis Zane. Satisfiability coding lemma. *Chicago J. Theoret. Comput. Sci.*, pages Article 11, 19 pp. (electronic), 1999.

10 Daniel Rolf. Improved Bound for the PPSZ/Schöning-Algorithm for 3-SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 1:111–122, 2006.

11 Uwe Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *40th Annual Symposium on Foundations of Computer Science (New York, 1999)*, pages 410–414. IEEE Computer Soc., Los Alamitos, CA, 1999.

12 Emo Welzl. Boolean satisfiability – combinatorics and algorithms (lecture notes), 2005. `http://www.inf.ethz.ch/~emo/SmallPieces/SAT.ps`.

The Complexity of Weighted Boolean #CSP Modulo k

Heng Guo[1], Sangxia Huang[2], Pinyan Lu[3], and Mingji Xia[4]

1 **University of Wisconsin-Madison**
Madison, WI 53706, USA
hguo@cs.wisc.edu

2 **KTH — Royal Institute of Technology**
Stockholm, Sweden
sangxia@csc.kth.se

3 **Microsoft Research Asia**
Beijing, China
pinyanl@microsoft.com

4 **Institute of Software, Chinese Academy of Sciences**
Beijing 100190, China
xmjljx@gmail.com

Abstract

We prove a complexity dichotomy theorem for counting weighted Boolean CSP modulo k for any positive integer $k > 1$. This generalizes a theorem by Faben for the unweighted setting. In the weighted setting, there are new interesting tractable problems. We first prove a dichotomy theorem for the finite field case where k is a prime. It turns out that the dichotomy theorem for the finite field is very similar to the one for the complex weighted Boolean #CSP, found by [Cai, Lu and Xia, STOC 2009]. Then we further extend the result to an arbitrary integer k.

1998 ACM Subject Classification F.2 [Theory of Computation] Analysis of Algorithms and Problem Complexity

Keywords and phrases #CSP, dichotomy theorem, counting problems, computational complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.249

1 Introduction

The complexity of counting problems is a fascinating subject. Valiant defined the class #P to capture most of these counting problems [21]. Several other related complexity classes are also well studied. One example is the $\oplus$P class, which consists of language L where there is a polynomial time nondeterministic Turing machine that on input $x \in L$ has an odd number of accepting computations, and on input $x \notin L$ has an even number of accepting computations [20, 18]. This class $\oplus$P can also be formulated as computing the parity of counting problems. In general, for any integer k, we may consider the counting problems modulo k, and the corresponding complexity class is denoted by $\#_k$P. The class $\oplus$P is in fact $\#_2$P.

Beyond the complexity of individual problems, there has been a great deal of interest in finding complexity dichotomy theorems which state that for a wide class of counting problems, every problem in the class is either computable in polynomial time (tractable) or hard (either NP-hard or #P-hard) [13, 12, 6, 15]. Such dichotomies do not hold without restrictions [17], assuming that the larger complexity class strictly contains P. The restrictions for which

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 249–260
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

dichotomy theorems are known can be framed in terms of local constraints, most importantly, Constraint Satisfaction Problems (CSP) [19, 10, 3, 4, 5, 11, 9, 14]. In this paper we address weighted #CSP problems, modulo any integer k, denoted by $\#_k$CSP.

Here we give a brief description of $\#_k$CSP. Let $\mathscr{F}$ be a set of functions, where each $F \in \mathscr{F}$ is a function mapping Boolean variables to a value. The weighted #CSP problem #CSP($\mathscr{F}$) is defined as follows: The input is a finite set of constraints on Boolean variables $x_1, x_2, \ldots, x_n$ of the form $F(x_{i_1}, x_{i_2}, \ldots, x_{i_k})$, where $F \in \mathscr{F}$. The output is

$$\sum_{x_1,x_2,\ldots,x_n \in \{0,1\}} \prod_{F \in \mathscr{F}} F(x_{i_1}, x_{i_2}, \ldots, x_{i_k}).$$

If each F takes values $0, 1$, then this counts the number of assignments "satisfying" all the Boolean constraints. Generally speaking, functions $F \in \mathscr{F}$ could take arbitrary values. What we consider, $\#_k$CSP, is the case that all F take integer values and the output is computed modulo k.

For #CSP, the complexity dichotomy theorem was first obtained for the unweighted case [10], and was later generalized to non-negative values [11]. Cai, Lu and Xia proved a dichotomy theorem for Boolean #CSP, where functions $F \in \mathscr{F}$ take arbitrary complex values [8]. Independently, a dichotomy theorem for real weighted functions was also obtained[2]. In these proofs, there are three extensively used reduction techniques: (1) Gadget construction, (2) polynomial interpolation, and (3) holographic transformation. As pointed out by Valiant [23], for finite fields, holographic transformations and interpolation both appear to offer less flexibility than they do for general counting problems.

There do exist several problems for which counting the number of solutions is #P-complete whereas computing it modulo some integer k is polynomial time computable. One prime example is computing the permanent of a 0/1 matrix, which is #P-complete [20]. The parity version of this problem corresponds to computing the permanent modulo 2, which is the same as the determinant modulo 2, and is therefore computable in polynomial time via linear algebra computations. Some more such tractable parity problems were recently given by Valiant [23]. Furthermore, the characteristic of the finite field may affect the tractability. For example, Valiant showed that $\#_7$Pl-Rtw-Mon-3CNF (counting the number of satisfying assignments of a planar read-twice monotone 3CNF formula, modulo 7) is solvable in P by a holographic algorithm [22], while the parity or general version of the same problem is $\oplus$P-hard or #P-hard, respectively.

These two facts (some useful techniques cannot be adopted in finite fields and there exist some more complicated tractable cases) make it more challenging to obtain a dichotomy for $\#_k$CSP problems. In [14], Faben obtained a dichotomy theorem for unweighted $\#_k$CSP. Essentially, there is no additional tractable case in his dichotomy theorem (except one obvious case). However, when we allow functions to take weights in the ring $\mathbb{Z}_k$, some new non-trivial tractable cases do emerge, which is similar to weighted vs unweighted #CSP without modulus. When moving from unweighted to real or complex weighted cases, the presence of both positive and negative values, and more generally, complex numbers, offers the opportunity of interesting cancelations, which could lead to efficient algorithms. In all such dichotomy theorems, roots of unity plays an essential roles [15, 8, 2, 7]. In finite fields, interesting cancelations do appear and every nonzero element is a root of unity. For general k, which may not be a prime, another subtlety is that the computation is performed in a ring $\mathbb{Z}_k$ rather than a field, where some nice property of a field no longer holds.

Our result starts from the finite field case, where the modulus k is an odd prime. In this case, the final result is algebraically the same as the dichotomy for complex weighted #CSP.

The imaginary unit $i = \sqrt{-1}$ plays an important role in the dichotomy for the complex weighted #CSP [8]. Here by "algebraically", we mean that we view i as a fourth primitive root of unit which is also well defined in a finite field (or its extension). Then the dichotomy for $\#_k$CSP is identical to that for complex weighted #CSP. Some of the proof techniques are fairly similar to those in the proof for the complex weighted case [8], while others are completely different. For example, the polynomial interpolation is one of the most important techniques in [8], but it is not available for the finite field.

Hereby we briefly explain why the polynomial interpolation does not work. Consider the simplest case where one would like to realize a unary function $[1, x]$ by polynomial interpolation. The answer to an instance of $\#_p$CSP including $[1, x]$, is a polynomial in the variable x. The degree of this polynomial is the number of occurrences of this function $[1, x]$. After replacing all of its occurrences by some realizable unary functions, we can evaluate the polynomial in other points of the variable. Given enough such evaluations, we can get a system of linear equations in the coefficients. The hope is to recover all coefficients by solving this system as long as its not singular. Then we can evaluate the polynomial in the original point x. In finite field $\mathbb{Z}_p$, we can reduce the degree of the polynomial to $p-1$ by Fermat's Little Theorem. So we have p different coefficients to recover. To get a non-singular linear system, we need to evaluate the polynomial on at least p points, which means we need to construct at least p different unary functions. However, in $\mathbb{Z}_p$, there are only p essentially different unary functions of the form $[1, x]$, and thus the interpolation is not even needed if we could construct all of them!

Another difference between the proof here and the one in [8] is that the norm of a complex number is used in [8]. This is an analytical, rather than algebraical, property of complex numbers, and is not available at all in finite fields. Such kinds of similarity and difference between fields with characteristic zero and finite p is one main theme of algebraical geometry [16]. It is interesting to observe similar phenomena in the complexity theory.

For general k, let $k = p_1^{r_1} p_2^{r_2} \cdots p_m^{r_m}$, where p_i's are distinct primes, be the prime factorization of k. By the Chinese Remainder Theorem, to solve the problem of $\#_k\mathrm{CSP}(\mathscr{F})$ is equivalent to solving all the $\#_{p_i^{r_i}}\mathrm{CSP}(\mathscr{F})$. For $\#_{p^r}$CSP and p being an odd prime, we prove a surprising result that $\#_{p^r}\mathrm{CSP}(\mathscr{F})$ is tractable iff $\#_p\mathrm{CSP}(\mathscr{F})$ is, assuming #P is not equal to P. One direction is trivial, namely if $\#_{p^r}\mathrm{CSP}(\mathscr{F})$ can be solved in polynomial time, so can $\#_p\mathrm{CSP}(\mathscr{F})$. The reduction in the other direction is not of the black box style. We need the dichotomy for $\#_p\mathrm{CSP}(\mathscr{F})$ to state all the tractable cases, assuming #P is not equal to P, and we also need to explicitly use algorithms to solve such tractable cases. The algorithm for $\#_{p^r}\mathrm{CSP}(\mathscr{F})$ has a time complexity which is n^r times larger than that of the algorithm for $\#_p\mathrm{CSP}(\mathscr{F})$. We use a different treatment to solve the case that $p = 2$.

2 Preliminaries

Let k be a given constant integer. In this paper we address the following type of counting problems, called weighted Boolean $\#_{\mathrm{k}}$CSP. Let $\mathscr{F}$ be a set of functions, where each $f \in \mathscr{F}$ is a function $f : \{0,1\}^r \to \mathbb{Z}$, mapping Boolean variables to integers. We call r the *arity* of f. The problem $\#_{\mathrm{k}}\mathrm{CSP}(\mathscr{F})$ is defined as follows: The input is a finite set of constraints on Boolean variables $x_1, x_2, \ldots, x_n$ of the form $f_j(x_{i_{j,1}}, x_{i_{j,2}}, \ldots, x_{i_{j,r_j}})$, where $f_j \in \mathscr{F}$. The output is

$$\left(\sum_{x_1, x_2, \ldots, x_n \in \{0,1\}} \prod_j f_j(x_{i_{j,1}}, x_{i_{j,2}}, \ldots, x_{i_{j,r_j}}) \right) \bmod k. \tag{1}$$

Since we are only interested in the final value modulus k, it is equivalent to view that all the functions take values in the ring $\mathbb{Z}_k$.

A *symmetric* function f of arity r on Boolean variables can be expressed by $[f_0, f_1, \ldots, f_r]$, where f_j is the value of f on inputs of weight j. We also use Δ_0, Δ_1 to denote $[1,0]$ and $[0,1]$ respectively. A binary function f is also expressed by the matrix $\begin{bmatrix} f(0,0) & f(0,1) \\ f(1,0) & f(1,1) \end{bmatrix}$.

Suppose f is a function on input variables $x_1, x_2, \ldots, x_r$. $f^{x_s=c}$ denotes the function $f^{x_s=c}(x_1, \ldots, x_{s-1}, x_{s+1}, \ldots, x_r) = f(x_1, \ldots, x_{s-1}, c, x_{s+1}, \ldots, x_r)$, and $f^{x_s=*}$ denotes the function $f^{x_s=*} = f^{x_s=0} + f^{x_s=1}$.

The underlying relation of f is given by $R_f = \{X \in \{0,1\}^r | f(X) \neq 0\}$. We also view relations as functions from $\{0,1\}^r$ to $\{0,1\}$. In this way, R_f could be viewed as *the unweighted version* of f. If the modulus k is a prime, we could copy f $k-1$ times to get f^{k-1} which, by Fermat's Little Theorem, is the unweighted version of it. In this way, we would be able to use some existing results for unweighted $\#_{\mathrm{k}}$CSP problems.

A relation $R \subseteq \{0,1\}^r$ being *affine* means it is the affine linear subspace composed of solutions of a system of affine linear equations, equivalently, if $a, b, c \in R$, then $a \oplus b \oplus c \in R$. If R_f is affine, we say f has affine support.

One important starting point of our work are the hardness results for unweighted $\#_{\mathrm{k}}$CSP [14]. For the unweighted case, every function f takes 1 if the input $X \in R_f$, and takes 0 otherwise.

▶ **Theorem 1.** [14] *Given an unweighted function set $\mathscr{F}$, and an integer k, $\#_{\mathrm{k}}\mathrm{CSP}(\mathscr{F})$ is computable in polynomial time if all the relations in $\mathscr{F}$ are affine, or if $k = 2$ and all functions in $\mathscr{F}$ are closed under complement. Otherwise it is $\#_k$P-hard* [1].

As a corollary, we have the following hardness result.

▶ **Corollary 2.** $\#_{\mathrm{k}}\mathrm{CSP}(\{[0,1,1]\})$ *and* $\#_{\mathrm{k}}\mathrm{CSP}(\{[1,1,0]\})$ *are $\#_k$P-hard for all k.*

We also need the following Pinning Lemma for $\#_{\mathrm{k}}$CSP.

▶ **Lemma 3.** *For every $\mathscr{F}$ and odd prime k, $\#_{\mathrm{k}}\mathrm{CSP}(\mathscr{F} \cup \{[1,0],[0,1]\}) \leq_T \#_{\mathrm{k}}\mathrm{CSP}(\mathscr{F})$.*

The proof is similar to that in [2].

We regard a function f and $c \cdot f$ as the same function, where c is a constant relatively prime to the modulus k. Our study on $\#_{\mathrm{k}}\mathrm{CSP}(\mathscr{F})$ starts with prime modulus. Doing computation modulo a prime is similar to computing with complex numbers in many aspects. For a given k, we define i_k as an element that satisfies $i_k^2 \equiv -1 \pmod{k}$. In some circumstances, i_k is an element of $\mathbb{Z}_k$, while in other situations, we need to extend the field and consider $\mathbb{Z}_k[x]/(x^2+1)$. There are essentially two elements satisfying this property, but it doesn't matter which one we pick as i_k.

We further define two classes of functions, for which the $\#_{\mathrm{k}}$CSP problems are tractable. Let X be an $r+1$ dimensional column vector $(x_1, x_2, \ldots, x_r, 1)$ over Boolean field $\mathbb{F}_2$. Suppose A is a Boolean matrix. χ_{AX} denotes the affine relation on inputs $x_1, x_2, \ldots, x_r$, whose value is 1 if AX is the zero vector, 0 if AX is not the zero vector.

$\mathscr{A}_k$ denotes all functions which have the form $\chi_{AX} i_k^{L_1(X)+L_2(X)+\cdots+L_n(X)}$ in modulo k, where L_j is a 0-1 indicator function $\chi_{\langle \alpha_j, X\rangle}$, α_j is a $r+1$ dimensional vector, and the inner

[1] We keep the statement as in Faben's paper. Technically, in the case that $k = 2k'$ where $k' > 1$ is odd, and $\mathscr{F}$ are closed under complement and not all affine, we believe that we can only claim the problem is $\#_{k'}$P-hard.

product $\langle \cdot, \cdot \rangle$ is over $\mathbb{Z}_2$. The additions among $L_j(X)$ are just the usual addition in $\mathbb{Z}$. Since it is the power of i, it can be computed modulo 4, but not modulo 2. (Since we ignore the global constant, all functions that are constant multiples of these functions are also in this class.)

$\mathscr{P}_k$ denotes the class of functions which, in modulo k, can be expressed as a product of unary functions, binary equality function ($[1, 0, 1]$), and binary disequality function ($[0, 1, 0]$).

It is often useful to view a $\#_{\mathrm{k}}$CSP instance as a bipartite graph $G = (U, V, E)$ where U corresponds to the set of constraints and V corresponds to the set of variables. Edge $(u, v) \in E$ iff variable v appears in constraint u. A subgraph of G is simply a certain combination of constraints in terms of CSP, and is sometimes called gadget. It is easy to see that if there are several connected components in G, then the result of the whole instance is exactly the product of that in the connected components. Therefore, it is sufficient to consider connected $\#_{\mathrm{k}}$CSP instances.

We also need some knowledge from number theory to deal with prime powers. Given k, a is a quadratic residue modulo k if there exists y such that $y^2 \equiv a \pmod{k}$. Thus, i_k exists in $\mathbb{Z}_k$ if and only if -1 is a quadratic residue modulo k.

▶ **Lemma 4.** *Let p be an odd prime and $k = p^r$. -1 is a quadratic residue modulo p if and only if it is a quadratic residue modulo k.*

Proof. The "if" direction is obvious. If there exist some $i_p \in [p]$ that $i_p^2 \equiv -1 \pmod{p}$, we consider the number $j_t = i_p + tp$, where t is an integer ranging from 0 to $p^{r-1} - 1$. If there exist t and t' that $j_t^2 \equiv j_{t'}^2 \pmod{p^r}$, it is easy to compute that $p^{r-1} | (p(t+t') + 2i_p)(t-t')$, because $|t - t'| < p^{r-1}$, $p | p(t+t') + 2i_p$. We get $p | 2i_p$, which is impossible. Thus the p^{r-1} values $\{j_t^2 \pmod{p^r}\}$ are distinct and there must exist some t such that $j_t^2 \equiv -1 \pmod{p^r}$. ◀

3 Complexity in the finite field $\mathbb{Z}_p$

In this section, we deal with the complexity of counting CSP problems in the finite field $\mathbb{Z}_p$. The parity case that $p = 2$, which is in fact the same as the unweighted case, has been solved in [14] (Theorem 1). In the following we always assume that p is an odd prime. All computations are done in the finite field $\mathbb{Z}_p$. For convenience, we often use the usual notation $=$ instead of $\equiv \pmod{p}$.

The counting CSP problem for $\mathscr{P}_p$ or $\mathscr{A}_p$ is tractable. The algorithm for $\mathscr{P}_p$ is based on decomposing functions into separated components that is easy to solve. The algorithm for $\mathscr{A}_p$ is similar to that for the complex weighted #CSP problems. We need the following two lemmas. The proof for the modulo case here is similar to the complex weighted case in [8].

▶ **Lemma 5.** *Let $F(x_1, x_2, \ldots, x_k) = \chi_{AX} i^{L_1(X)+L_2(X)+\cdots+L_n(X)} \in \mathscr{A}$. If $AX = 0$ is infeasible over $\mathbb{Z}_2$, then $\sum_{x_1,x_2,\ldots,x_k} F = 0$. If $AX = 0$ is feasible, then in polynomial time, we can construct another function $H(y_1, y_2, \ldots, y_s) = i^{L'_1(Y)+L'_2(Y)+\cdots+L'_n(Y)} \in \mathscr{A}$, such that $0 \le s \le k$, and $\sum_{x_1,x_2,\ldots,x_k} F = \sum_{y_1,y_2,\ldots,y_s} H$.*

▶ **Lemma 6.** *Let $F(x_1, x_2, \ldots, x_k) = i^{L_1(X)+L_2(X)+\cdots+L_n(X)}$. Exactly one of the following two statements hold:*

1. (Congruity) *There exists a constant $c \in \{1, -1, i, -i\}$ such that for all $x_2, x_3, \ldots, x_k \in \{0, 1\}$ we have $F^{x_1=1}/F^{x_1=0}(x_2, x_3, \ldots, x_k) = c$;*
2. (Semi-congruity) *There exists a constant $c \in \{1, i\}$ and an affine subspace S of dimension $k - 2$ on $T = \{(x_2, x_3, \ldots, x_k) \mid x_i \in \mathbb{Z}_2\}$, such that $F^{x_1=1}/F^{x_1=0}(x_2, x_3, \ldots, x_k) = c$ on S, and $F^{x_1=1}/F^{x_1=0}(x_2, x_3, \ldots, x_k) = -c$ on $T - S$.*

We first apply Lemma 5 to get rid of the χ_{AX} factor, and then use the congruity or semi-congruity property of constraint functions proved in Lemma 6 to eliminate variables one by one. Details can be found in [8].

We will then show that for any other functions the problem is hard. Before the hardness proof we will mention two constructions, which will be used throughout our proofs.

Given a function f and any positive integer k, we can simulate a function g such that any entry of g is the corresponding entry of f to the kth power. This is done by connecting corresponding edges of k copies of f with an equality of arity $k+1$. Figure 1 is a simple example when $k=2$ and the arity of f is 3. The other construction is for binary functions. Given two binary functions f and g, whose matrices are F and G respectively, we connect them directly via a binary equality as shown in Figure 2. It is easy to check that the matrix of the resulting function is FG.

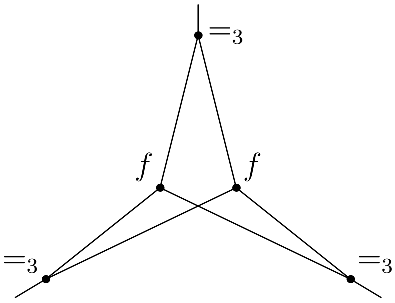


Figure 1 Duplicate two copies of f

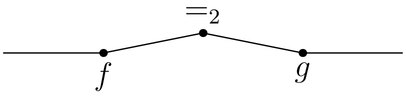


Figure 2 Directly connect two binary functions

The starting point of our hardness proof is the following lemma. In the rest of this section we may omit the subscripts of $\mathscr{A}_p$ and $\mathscr{P}_p$ when it is clear from context.

▶ **Lemma 7.** *If $[a,b,c] \notin \mathscr{A} \cup \mathscr{P}$, $\#_{\mathrm{p}}\mathrm{CSP}(\{[a,b,c]\})$ is $\#_p$P-hard. To be explicit, all tractable functions $[a,b,c]$ from $\mathscr{A} \cup \mathscr{P}$ have one of the following forms: $[x,0,y]$, $[0,x,0]$, $[x^2,xy,y^2]$, $x[1,\pm i,1]$ or $x[1,\pm 1,-1]$.*

This lemma says, if restricted to one single symmetric binary function, a dichotomy theorem holds. The same lemma also served as the hardness starting point for the complex weighted dichotomy [8]. However, the proof techniques are completely different. The main proof tool for the complex weighted dichotomy [8] is polynomial interpolation, which is not available here as was explained in Section 1. Before proving this lemma, we state several useful facts.

▶ **Lemma 8.** *For any symmetric binary function $[0,b,c]$ and a prime number p, where $bc \not\equiv 0 \pmod{p}$, $\#_{\mathrm{p}}\mathrm{CSP}(\{[0,b,c]\})$ is $\#_p$P-hard.*

Proof. Via the construction mentioned above, we can realize $[0,b^k,c^k]$. Taking $k=p-1$, by Fermat's Little Theorem, it becomes $[0,1,1]$. By Corollary 2, the $\#_p$CSP problem is $\#_p$P-hard. ◀

We also need the following lemma to realize new binary functions. The new binary functions are not by an explicit construction but an existent argument, which crucially uses the finiteness of the field.

▶ **Lemma 9.** *For any non-degenerate* 2×2 *matrix* A *in* $\mathbb{Z}_p$*, there exists* k *such that* $A^k \equiv I \pmod{p}$ *where* I *is the identity matrix.*

Proof. Since $\mathbb{Z}_p$ is finite, there are finitely many non-degenerate 2×2 matrices. Thus, by Pigeonhole Principle, there exists p and q such that $p < q$ and $A^p \equiv A^q \pmod{p}$. Taking the smallest such pair and letting $k = q - p$, it is easy to see that $A^k \equiv I \pmod{p}$. ◀

▶ **Corollary 10.** *For any non-degenerate* 2×2 *matrix* A *in* $\mathbb{Z}_p$*, there exists a positive integer* k *such that* $A^k \equiv A^{-1} \pmod{p}$*.*

▶ **Lemma 11.** *Let* F *be a function of matrix* $\begin{bmatrix} a & b \\ c & d \end{bmatrix}$*, where* $p \nmid abcd$ *and* $a^2d^2 \not\equiv b^2c^2 \pmod{p}$*.* $\#_{\mathrm{p}}\mathrm{CSP}(\{F\})$ *is* $\#_p$*P-hard.*

Proof. We can realize $\begin{bmatrix} a^2 & c^2 \\ b^2 & d^2 \end{bmatrix}$ by two copies of the function. Since $a^2d^2 \neq b^2c^2$, this matrix is non-degenerate. Thus by Corollary 10, we can realize $(a^2d^2 - b^2c^2)^{-1} \begin{bmatrix} d^2 & -c^2 \\ -b^2 & a^2 \end{bmatrix}$. As we consider the problem in the field $\mathbb{Z}_p$, $(a^2d^2 - b^2c^2)^{-1}$ is just a constant factor and we may ignore it. By the pinning Lemma 3, we can realize $[d^2, -c^2]$, and hence $[d^2, 0, -c^2]$, by connecting it to a $=_3$. Then the following function

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} d^2 & 0 \\ 0 & -c^2 \end{bmatrix} \begin{bmatrix} a & c \\ b & d \end{bmatrix} = \begin{bmatrix} a^2d^2 - b^2c^2 & acd^2 - bc^2d \\ acd^2 - bc^2d & 0 \end{bmatrix},$$

or $[a^2d^2 - b^2c^2, cd(ad - bc), 0]$ is realizable. Since $p \nmid abcd$ and $a^2d^2 \neq b^2c^2$, we have $a^2d^2 - b^2c^2 \neq 0$ and $cd(ad - bc) \neq 0$. By Lemma 8, $\#_{\mathrm{p}}\mathrm{CSP}(\{F\})$ is $\#_p$P-hard. ◀

Now we can prove Lemma 7.

Proof. (Lemma 7) If $a = 0$, we know $bc \neq 0$, otherwise it is in one of the five exceptional cases. So by Lemma 8, $\#_{\mathrm{p}}\mathrm{CSP}(\{[a, b, c]\})$ is $\#_p$P-hard . The case $c = 0$ is symmetric. Since $[a, b, c] \notin \mathscr{A} \cup \mathscr{P}$, we know $b \neq 0$. Therefore we will assume in the following that $abc \neq 0$.

By Lemma 11, $\#_{\mathrm{p}}\mathrm{CSP}(\{[a, b, c]\})$ is $\#_p$P-hard if $b^4 \neq a^2c^2$. Moreover, if $b^2 = ac$, then $[a, b, c] \in \mathscr{P}$. Therefore in the following, we assume that $b^2 = -ac$. Since $[a, b, c] \notin \mathscr{A} \cup \mathscr{P}$, we must have that $a \neq \pm c$.

Next we connect two copies of $[a, b, c]$ to realize $[a^2 + b^2, ab + bc, b^2 + c^2]$. Since $b^2 = -ac$, we actually have $[a(a - c), b(a + c), c(c - a)] \triangleq [a', b', c']$. It is easy to verify that $b'^4 \neq a'^2c'^2$, and thus $\#_{\mathrm{p}}\mathrm{CSP}(\{[a, b, c]\}) \geq_T \#_{\mathrm{p}}\mathrm{CSP}(\{[a', b', c']\})$, which is $\#_p$P-hard . ◀

▶ **Lemma 12.** *If* R_F *is not affine, then* $\#_{\mathrm{p}}\mathrm{CSP}(\{F\})$ *is* $\#_p$*P-hard.*

Proof. We can easily reduce the unweighted case to the weighted one, the hardness follows. ◀

Now we come to the two key lemmas for the hardness proof. Both proofs inductively reduce the arity of a function. Suppose $\mathscr{F} \not\subseteq \mathscr{A}$ and $\mathscr{F} \not\subseteq \mathscr{P}$. Thus there exists $F \notin \mathscr{A}$ and $G \notin \mathscr{P}$, where $F, G \in \mathscr{F}$. (It is possible that $G = F$). From F and G, we recursively simulate functions with smaller arity, keeping the property of being not in $\mathscr{A}$ and not in $\mathscr{P}$ respectively. The proofs of these two lemmas are very similar to those in [8]. Due to space limitation, we give a sketch of proof for Lemma 14 in the Appendix and omit the proof for Lemma 13.

▶ **Lemma 13.** *If $F \notin \mathscr{A}$, then either $\#_{\mathrm{p}}\mathrm{CSP}(\{F\})$ is $\#_p$P-hard, or we can simulate a unary function $H \notin \mathscr{A}$, that is, there is a reduction from $\#_{\mathrm{p}}\mathrm{CSP}(\{F, H\})$ to $\#_{\mathrm{p}}\mathrm{CSP}(\{F\})$.*

▶ **Lemma 14.** *For any function $F \notin \mathscr{P}$, either $\#_{\mathrm{p}}\mathrm{CSP}(\{F\})$ is $\#_p$P-hard , or we can simulate, using F, a function $[a, 0, 1, 0]$ (or $[0, 1, 0, a]$), where $a \neq 0$, or a binary function $H \notin \mathscr{P}$ having no zero values.*

Now we are ready to prove the main lemma.

▶ **Lemma 15.** *Let p be an odd prime, and $\mathscr{F}$ a class of functions mapping Boolean inputs to $[p]$. If $\mathscr{F} \subseteq \mathscr{A}$ or $\mathscr{F} \subseteq \mathscr{P}$, $\#_{\mathrm{p}}\mathrm{CSP}(\mathscr{F})$ is computable in polynomial time. Otherwise, $\#_{\mathrm{p}}\mathrm{CSP}(\mathscr{F})$ is $\#_p$P-hard .*

Proof. For $\mathscr{A}$ and $\mathscr{P}$, their polynomial time algorithms are given above.

If $\mathscr{F} \nsubseteq \mathscr{A}$ and $\mathscr{F} \nsubseteq \mathscr{P}$, by Lemma 13, either $\#_{\mathrm{p}}\mathrm{CSP}(\mathscr{F})$ is $\#_p$P-hard , or we can simulate a function $F = [1, \lambda] \notin \mathscr{A}$. In particular $\lambda \notin \{0, \pm 1, \pm i\}$. By Lemma 14, either $\#_{\mathrm{p}}\mathrm{CSP}(\mathscr{F})$ is $\#_p$P-hard , or we can simulate a function $P = [a, 0, 1, 0]$, or $P' = [0, 1, 0, a]$, where $a \neq 0$, or a binary function $H \notin \mathscr{P}$ having no zero values.

Firstly, we prove $\#_{\mathrm{p}}\mathrm{CSP}(\{F, P\})$ is $\#_p$P-hard . Clearly $P^{x_1=*} = [a, 1, 1]$. If $a \notin \{1, -1\}$, it is $\#_p$P-hard by Lemma 7. If $a \in \{1, -1\}$, we can construct $Q(x_1, x_2) = \sum_{x_3} P(x_1, x_2, x_3)F(x_3) = [a, \lambda, 1]$, which is $[\pm 1, \lambda, 1]$. Both of them are $\#_p$P-hard by Lemma 7. The proof for $\#_{\mathrm{p}}\mathrm{CSP}(\{F, P'\})$ is the same.

Secondly, we prove $\#_{\mathrm{p}}\mathrm{CSP}(\{F, H\})$ is $\#_p$P-hard . After normalizing, we may suppose $H = \begin{bmatrix} 1 & x \\ y & z \end{bmatrix}$, where $xyz \neq 0$, and $z \neq xy$. There are two cases, depending on whether $z = -xy$.

For the case $z \neq -xy$, we conclude that it is hard by applying Lemma 11 on H.

For the case $z = -xy$, we construct some binary functions with an integer parameter s as follows:

$$\begin{aligned} \sum_{x_3} H(x_1, x_3)H(x_2, x_3)(F(x_3))^s &= [1 + \lambda^s x^2, (y + \lambda^s xz), (y^2 + \lambda^s z^2)] \\ &= [1 + \lambda^s x^2, y(1 - \lambda^s x^2), y^2(1 + \lambda^s x^2)]. \end{aligned}$$

As λ is not a power of i, at most one of the two values x^2 and λx^2 can be a power of i. Now we choose $s = 0$ or $s = 1$ above so that $\lambda^s x^2 \notin \{\pm 1, \pm i\}$.

After normalizing, we may write the function $[1 + \lambda^s x^2, y(1 - \lambda^s x^2), y^2(1 + \lambda^s x^2)]$ as $[1, y(1 - \lambda^s x^2)(1 + \lambda^s x^2)^{-1}, y^2]$, noticing that $1 + \lambda^s x^2 \neq 0$. We claim that this function is not one of the five tractable cases from Lemma 7. Since there are no zero entries, clearly it is not the first two cases. It has rank 2, therefore it is not the third case. If it were the fourth tractable case $[1, \pm i, 1]$, then $y = \pm 1$, and $(1 - \lambda^s x^2)(1 + \lambda^s x^2)^{-1} = \pm i$. This implies that $\lambda^s x^2 = \pm i$, which is impossible. If $[1, y(1 - \lambda^s x^2)(1 + \lambda^s x^2)^{-1}, y^2] = [1, \pm 1, -1]$, the fifth tractable case, then $y = \pm i$, and again $(1 - \lambda^s x^2)(1 + \lambda^s x^2)^{-1} = \pm i$, also impossible. ◀

4 Dichotomy for a general integer k

We first deal with the case when k is a power of an odd prime. Then we use Chinese Remainder Theorem to prove hardness for other composite numbers. Beigel and Gill have shown that the class $\#_{p^r}P$ is the same as $\#_p P$ and for a composite k having two or more prime factors, the modulo counting class is a union of counting classes modulo each of its prime factors [1]. Therefore, we only talk about $\#_p$P-hard for prime p.

▶ **Lemma 16.** *Suppose p is an odd prime, $k = p^r$ for some integer $r \geq 1$. $\mathscr{F}$ is a finite set of constraint functions. If, after taking modulus p, $\mathscr{F} \subseteq \mathscr{A}_p$ or $\mathscr{F} \subseteq \mathscr{P}_p$, $\#_{\mathrm{k}}\mathrm{CSP}(\mathscr{F})$ has polynomial time algorithms. Otherwise, $\#_{\mathrm{k}}\mathrm{CSP}(\mathscr{F})$ is $\#_p$P-hard .*

Proof. If $\#_{\mathrm{p}}\mathrm{CSP}(\mathscr{F})$ is $\#_p$P-hard, then $\#_{\mathrm{k}}\mathrm{CSP}(\mathscr{F})$ must be $\#_p$P-hard. Therefore we only need to show the tractable part. We decompose every function $f \in \mathscr{F}$ into the sum of two functions g_f and h_f, such that all values of h_f are multiples of p, and $g_f \in \mathscr{P}_k$ or $g_f \in \mathscr{A}_k$, depending on whether $f \in \mathscr{P}_p$ or $f \in \mathscr{A}_p$. Such a decomposition is always possible. If $f \in \mathscr{P}_p$, assuming that $f = \prod f_i$, where f_i is either unary, or binary equality or disequality, the we can simply take $g_f = \prod f_i$ in modulo k. Since $g_f \equiv f \pmod{p}$, we can see that values of $h_f = f - g_f$ are multiples of p. On the other hand, if $f \in \mathscr{A}_p$, then f could be expressed as $f = \chi_{AX} i_p^{\sum L_i(X)}$. Let $g_f = \chi_{AX} i_k^{\sum L_i(X)}$. It is easy to see that $g_f \equiv f \pmod{p}$, and h_f satisfies our condition.

Given the decomposition, we can express the final summation in the following way

$$\begin{aligned} & \sum_{x_1,x_2,\ldots,x_n \in \{0,1\}} \prod f(x_{i_1}, x_{i_2}, \ldots, x_{i_r}) \bmod k \\ = & \sum_{x_1,x_2,\ldots,x_n \in \{0,1\}} \prod (g_f(x_{i_1}, x_{i_2}, \ldots, x_{i_r}) + h_f(x_{i_1}, x_{i_2}, \ldots, x_{i_r})) \bmod k \\ = & \sum_{f'_1 \in \{g_{f_1}, h_{f_1}\}} \cdots \sum_{f'_n \in \{g_{f_n}, h_{f_n}\}} \left(\sum_{x_1,x_2,\ldots,x_n \in \{0,1\}} \prod f'_i(x_{i_1}, x_{i_2}, \ldots, x_{i_r}) \right) \bmod k \end{aligned}$$

We only need to consider the assignments of functions such that the summation in the parenthesis is nonzero. To ensure that this is nonzero, no more than r of the functions f'_i can be assigned h_{f_i}. The total number of such combinations is of order $O(n^{r+1})$.

For every such combination, assume that $f'_1, \ldots, f'_r$ are assigned. Since $\mathscr{F}$ is finite, the degree of the functions is bounded. Therefore, constantly many variables are involved in f'_1, $\ldots$, f'_r. We can list all assignments to these variables in constant time. We can express each assignment with the help of Lemma 3, and obtain a new instance in $\#_{\mathrm{k}}\mathrm{CSP}(\mathscr{F}')$ such that $\mathscr{F}' \subseteq \mathscr{A}$ or $\mathscr{F}' \subseteq \mathscr{P}$, depending on the case of $\mathscr{F}$. Therefore, we can compute the value of these instances in polynomial time, and thus we can compute the value of the whole instance efficiently. ◀

Now we deal with $k = 2^r$. We need the following claim to establish the connection between the weighted and unweighted case.

▶ **Lemma 17.** *For any positive integer r and t, $(1+2t)^{2^r} \equiv 1 \pmod{2^r}$.*

Therefore, we only need to duplicate the weighted functions $k = 2^r$ times to obtain an unweighted function. Note this process actually converts all odd values to 1, and all even values to 0. Then we have the following result for $k = 2^r$.

▶ **Lemma 18.** *If $k = 2^r$ and $r > 1$, then $\#_{\mathrm{k}}\mathrm{CSP}(\mathscr{F})$ is $\#_2$P-hard, unless all functions in $\mathscr{F}$ are affine modulo 2, for which we have a polynomial time algorithm.*

Proof. Hardness can be proved by considering the unweighted version of $\mathscr{F}$ and applying Theorem 1. Algorithm for an affine $\mathscr{F}$ is similar to that in Lemma 16, except that for a given combination and assignment, we calculate the value of the gadget directly instead of applying the Pinning Lemma. This can be done efficiently according to [8]. ◀

Based on Lemma 15, Lemma 16, Lemma 18 and the Chinese Remainder Theorem, we conclude with our main result:

▶ **Theorem 19.** *Let $k = 2^{r_0} p_1^{r_1} p_2^{r_2} \cdots p_m^{r_m}$, where p_i's are distinct odd primes, $r_0 \geq 0$, and $r_i \geq 1$ for $i = 1, 2, ..., m$. Let $\mathscr{F}$ be a set of functions. $\#_k$CSP($\mathscr{F}$) is in P if one of the following three conditions is satisfied.*

1. *$r_0 = 0$. $\mathscr{F} \subseteq \mathscr{A}_{p_i}$ or $\mathscr{F} \subseteq \mathscr{P}_{p_i}$ for all $i \in [m]$.*
2. *$r_0 = 1$. $\mathscr{F} \subseteq \mathscr{A}_2$ or every function in $\mathscr{F}$ are closed under complement after mod 2. $\mathscr{F} \subseteq \mathscr{A}_{p_i}$ or $\mathscr{F} \subseteq \mathscr{P}_{p_i}$ for all $i \in [m]$.*
3. *$r_0 \geq 2$. $\mathscr{F} \subseteq \mathscr{A}_2$. $\mathscr{F} \subseteq \mathscr{A}_{p_i}$ or $\mathscr{F} \subseteq \mathscr{P}_{p_i}$ for all $i \in [m]$.*

Otherwise the problem is $\#_p$P-hard for some $p|k$. More specific, we have

- *For $i \in [m]$, if $\mathscr{F} \nsubseteq \mathscr{A}_{p_i}$ and $\mathscr{F} \nsubseteq \mathscr{P}_{p_i}$, then $\#_k$CSP($\mathscr{F}$) is $\#_{p_i}$P-hard.*
- *If $r_0 = 1$, $\mathscr{F} \nsubseteq \mathscr{A}_2$, and it is not the case that every function in $\mathscr{F}$ are closed under complement after mod 2, then $\#_k$CSP($\mathscr{F}$) is $\#_2$P-hard.*
- *If $r_0 \geq 2$ and $\mathscr{F} \nsubseteq \mathscr{A}_2$, then $\#_k$CSP($\mathscr{F}$) is $\#_2$P-hard.*

The statement of the main theory is a little complicated due to technique reason [2] . In terms of dichotomy, we have a simple statement.

▶ **Theorem 20.** *Let $k > 1$ and $\mathscr{F}$ be a set of functions. Then $\#_k$CSP($\mathscr{F}$) is either in P or $\#_p$P-hard for some $p|k$.*

Acknowledgements Work partly done when Heng Guo was a master student in Peking University, and Sangxia Huang was an intern student at Microsoft Research Asia and an undergraduate student of Shanghai Jiao Tong University. Sangxia Huang is supported by ERC Advanced investigator grant 226203. Mingji Xia is supported by NSFC 61003030 and 60970003, the CAS start-up fund for CAS President Scholarship winner, the Hundred-Talent program of Chinese Academy of Sciences under Angsheng Li, and the Grand Challenge Program "Network Algorithms and Digital Information" of ISCAS.

References

1 Richard Beigel and John Gill. Counting classes: Thresholds, parity, mods, and fewness. *Theor. Comput. Sci.*, 103(1):3–23, 1992.

2 A. Bulatov, M. Dyer, L.A. Goldberg, M. Jalsenius, and D. Richerby. The complexity of weighted boolean #CSP with mixed signs. *Theoretical Computer Science*, 410(38-40):3949–3961, 2009.

3 Andrei A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element set. *J. ACM*, 53(1):66–120, 2006.

4 Andrei A. Bulatov. The complexity of the counting constraint satisfaction problem. 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 646–661. Springer, 2008.

5 Andrei A. Bulatov and Víctor Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. In *FOCS*, pages 562–571. IEEE Computer Society, 2003.

6 Andrei A. Bulatov and Martin Grohe. The complexity of partition functions. In Josep Díaz, Juhani Karhumäki, Arto Lepistö, and Donald Sannella, editors, *ICALP*, volume 3142 of *Lecture Notes in Computer Science*, pages 294–306. Springer, 2004.

[2] The statement of our dichotomy is slightly different with Faben's [14]. The dichotomy by Faben stated that a problem is either P or $\#_k$P-hard. This is because in the unweighted case, the criterion for different odd prime p is identical.

7 Jin-Yi Cai, Xi Chen, and Pinyan Lu. Graph homomorphisms with complex values: A dichotomy theorem. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP (1)*, volume 6198 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2010.

8 Jin-Yi Cai, Pinyan Lu, and Mingji Xia. Holant problems and counting CSP. In Michael Mitzenmacher, editor, *STOC*, pages 715–724. ACM, 2009.

9 N. Creignou, S. Khanna, and M. Sudan. *Complexity classifications of boolean constraint satisfaction problems*. SIAM Monographs on Discrete Mathematics and Applications, 2001.

10 Nadia Creignou and Miki Hermann. Complexity of generalized satisfiability counting problems. *Inf. Comput.*, 125(1):1–12, 1996.

11 Martin E. Dyer, Leslie Ann Goldberg, and Mark Jerrum. The complexity of weighted boolean #CSP. *SIAM J. Comput.*, 38(5):1970–1986, 2009.

12 Martin E. Dyer, Leslie Ann Goldberg, and Mike Paterson. On counting homomorphisms to directed acyclic graphs. *J. ACM*, 54(6), 2007.

13 Martin E. Dyer and Catherine S. Greenhill. The complexity of counting graph homomorphisms. *Random Struct. Algorithms*, 17(3-4):260–289, 2000.

14 John Faben. The complexity of counting solutions to generalised satisfiability problems modulo k. *CoRR*, abs/0809.1836, 2008.

15 Leslie Ann Goldberg, Martin Grohe, Mark Jerrum, and Marc Thurley. A complexity dichotomy for partition functions with mixed signs. In *STACS*, pages 493–504, 2009.

16 R. Hartshorne. *Algebraic geometry*. springer Verlag, 1977.

17 Richard E. Ladner. On the structure of polynomial time reducibility. *J. ACM*, 22(1):155–171, 1975.

18 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, 1982.

19 T.J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, page 226. ACM, 1978.

20 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

21 Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

22 Leslie G. Valiant. Accidental algorthims. In *FOCS '06: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science*, pages 509–517, Washington, DC, USA, 2006. IEEE Computer Society.

23 Leslie G. Valiant. Some observations on holographic algorithms. In Alejandro López-Ortiz, editor, *LATIN*, volume 6034 of *Lecture Notes in Computer Science*, pages 577–590. Springer, 2010.

A Proof of Lemma 14

Proof. Suppose F has arity r. Since $\mathscr{P}$ contains all unary functions and $F \notin \mathscr{P}$, $r \geq 2$. Define an $|R_F| \times r$ $\{0,1\}$-matrix whose rows list every element of R_F, and columns correspond to $x_1, \ldots, x_r$.

We first remove any column which is all-0 or all-1 and update the table to $R_{F^{x_i=0}}$ or $R_{F^{x_i=1}}$, respectively. If two columns are identical or are complementary in every bit, we remove one of them and update the table to $R_{F^{x_j=*}}$, where j corresponds to the column removed. We remove columns as long as possible. It is easy to see that this removal process maintains the property of not belonging to $\mathscr{P}$.

Now we suppose there is some $G \notin \mathscr{P}$ where no more columns can be removed by the above process. There must be some columns left in the table, otherwise the function just before the last column removal is a unary function, hence in $\mathscr{P}$. In fact G being not in $\mathscr{P}$, the arity of G is ≥ 2. For simplicity we still denote it by r. We have two cases:

Case 1: $|R_G| < 2^r$. By Lemma 12, we may assume R_G is affine, given by an affine linear system $AX = 0$. We have shown that $|R_G| \neq 0$, as some columns remain. Since G is not unary, the table has more than one columns. If $|R_G| = 1$, any two columns (of length one) must be identical or complementary and the removal process should have continued. Thus $|R_G| > 1$. W.l.o.g. assume $x_1, \ldots, x_s$ are free variables in $AX = 0$ and $x_{s+1}, \ldots, x_k$ are dependent variables. $|R_G| = 2^s$ is a power of 2. We have shown that $s \geq 1$. By $|R_G| < 2^r$, $s < r$. We claim $s \geq 2$. If instead $s = 1$, then every $x_2, \ldots, x_r$ is dependent on x_1 on R_G, so the column at x_2 must be an all-0 or all-1 column, or be identical or complementary to x_1. The expression of x_r in terms of $x_1, \ldots, x_s$ must involve at least two non-zero coefficients; otherwise the column at x_r must be an all-0 or all-1 column, or be identical or complementary to another column. W.l.o.g., say the coefficients of x_1, x_2 are non-zero.

Let $P(x_1, x_2, x_r) = G^{x_3=0,\ldots,x_s=0,x_{s+1}=*,\ldots,x_{r-1}=*}$ (these two lists of variables could be empty). It can be verified that $R_P = \chi_{x_1 \oplus x_2 \oplus x_r = c}$ for some $c \in \mathbb{Z}_2$.

The affine linear equation $x_1 \oplus x_2 \oplus x_r = c$ is symmetric. Now we define a "symmetrized" function $H(x_1, x_2, x_r) = \prod_{\sigma \in S_3} P(x_{\sigma(1)}, x_{\sigma(2)}, x_{\sigma(r)})$, where S_3 is the symmetry group on three letters $\{1, 2, k\}$. This H is a symmetric function on (x_1, x_2, x_r) and has support $R_H = R_P$. Thus, after normalizing, $H = [a, 0, 1, 0]$ or $[0, 1, 0, a]$ where $a \neq 0$. We remark that this ternary function $H \notin \mathscr{P}$.

Case 2: $|R_G| = 2^r$. If for all $1 \leq i \leq r$, the ratio $G^{x_i=1}/G^{x_i=0}$ is a constant function c_i, (since $|R_G| = 2^r$ there are no divisions by zeros), then $G = c_0 \cdot \prod_{1 \leq i \leq r} U_i(c_i)$, where the constant $c_0 = G^{x_1=0,\ldots,x_r=0}$, and $U_i(c_i)$ is the unary function $[1, c_i]$ on x_i. This gives $G \in \mathscr{P}$, a contradiction.

Now suppose for some i, $G^{x_i=1}/G^{x_i=0}$ is not a constant function. W.l.o.g., assume that $i = 1$. The Boolean hypercube on $(x_2, \ldots, x_r) \in \{0, 1\}^{r-1}$ is connected by edges which flip just one bit. W.l.o.g., suppose that

$G^{x_1=1}/G^{x_1=0}(0, a_3, \ldots, a_r) \neq G^{x_1=1}/G^{x_1=0}(1, a_3, \ldots, a_r)$. Set $x_3 = a_3, \ldots, x_r = a_r$, we get a binary function $H(x_1, x_2) = G(x_1, x_2, a_3, \ldots, a_r)$. We have that $H(1, 0)/H(0, 0) \neq H(1, 1)/H(0, 1)$, hence the rank of $H = \begin{bmatrix} H(0,0) & H(0,1) \\ H(1,0) & H(1,1) \end{bmatrix}$ is 2.

If H were in $\mathscr{P}$, then partition the variable set according to connectivity by binary equality and disequality functions. If any connected component has at least 2 variables, we can set values to these 2 variables so that $H = 0$. But H is never zero. Then each component must be a single variable and H is defined by a product of unary functions. But such a function has rank 1. This contradiction completes our proof. ◀

The #CSP Dichotomy is Decidable

Martin Dyer[1] **and David Richerby***

[1] **School of Computing, University of Leeds, LS2 9JT, UK**

Abstract

Bulatov (2008) and Dyer and Richerby (2010) have established the following dichotomy for the counting constraint satisfaction problem (#CSP): for any constraint language Γ, the problem of computing the number of satisfying assignments to constraints drawn from Γ is either in FP or is #P-complete, depending on the structure of Γ. The principal question left open by this research was whether the criterion of the dichotomy is decidable. We show that it is; in fact, it is in NP.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Constraint satisfaction problem, counting problems, complexity dichotomy, decidability.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.261

1 Introduction

Many important and natural problems in areas such as graph theory, Boolean logic, databases, type inference, scheduling, artificial intelligence and even theoretical physics can be expressed naturally as constraint satisfaction problems (CSPs) [9, 13]. In such problems, we seek to assign values from some domain to variables, while simultaneously satisfying a collection of constraints on the values that may be taken by given combinations of the variables.

For example, graph three-colourability is the problem of deciding whether we can assign one of three colours (domain values) to each vertex of a graph (variables) such that no edge joins vertices with the same colour (constraints). Since it includes this well-known NP-complete problem, it is immediate that this general form of CSP, known as *uniform* CSP is, itself, NP-complete.

For this reason, attention has focused on the so-called *nonuniform* version of CSP. Here, we fix a domain D and a set Γ of relations over D, known as the *constraint language*. We write CSP(Γ) for the version of CSP where we only allow constraints of the form, "the values assigned to variables $v_1, \dots, v_r$ must form a tuple in the r-ary relation $H \in \Gamma$." Note that all the constraints needed to express three-colourability can be written by taking D to be any three-element set and letting Γ contain just the binary disequality relation on D.

It follows that, for some Γ, even the restricted problem CSP(Γ) is NP-complete. However, taking Γ to be the binary disequality relation on a two-element domain allows us to express graph two-colourability, which is in P. Feder and Vardi [13] conjectured that these are the only possibilities: that is, for all Γ, CSP(Γ) is in P or is NP-complete. To date, this conjecture remains open but it is known to hold in special cases [1, 14, 18]. Recent efforts to resolve the conjecture have focused on techniques from universal algebra [6].

It follows from Ladner's theorem that there can be no such dichotomy for the whole of NP, since either P = NP or there is an infinite, strict hierarchy of complexity classes between

* Supported by EPSRC grant EP/E062172/1 "The Complexity of Counting in Constraint Satisfaction Problems".

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 261–272
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

the two [15]. Therefore, if $\mathsf{P} \neq \mathsf{NP}$, there are problems in NP that are neither in P nor NP-complete. However, a dichotomy for CSP is still possible because the problems expressible as $\mathsf{CSP}(\Gamma)$ for some Γ are a proper subset of NP. In particular, there is no Γ such that $\mathsf{CSP}(\Gamma)$ defines graph Hamiltonicity or even graph connectivity (this follows from [11, 12]). Further, Ladner's proof is via a diagonalisation that does not seem to be expressible in CSP [13].

In the present paper, we consider the *counting constraint satisfaction problem*, #CSP. Here, we are interested in the number of satisfying assignments to CSP instances. For several restricted classes of constraint language Γ, it was known that #CSP(Γ) is either in polynomial time or #P-complete [4, 5, 7–9].

Bulatov successfully proved the dichotomy for all Γ [2,3], showing that $\#\mathsf{CSP}(\Gamma)$ is always either computable in polynomial time or #P-complete. He made extensive use of techniques from universal algebra; the present authors gave an elementary proof of an equivalent dichotomy [10]. The principal question left open by this research was the decidability of the distinct but equivalent criteria: that is, whether there is an algorithm that determines for which Γ $\#\mathsf{CSP}(\Gamma)$ is tractable and for which it is #P-complete. In this paper, we demonstrate such an algorithm.

We first describe the dichotomy — formal definitions will be given later. A ternary relation R is *balanced* if the matrix $M(x, y) = |\{z : xyz \in R\}|$ decomposes into blocks of rank one. A relation $R \subseteq D^r$ of arity $r \geq 3$ can be considered as a ternary relation over $D^k \times D^\ell \times D^{r-k-\ell}$ for any $k, \ell \geq 1$ with $k + \ell < r$. We say that R is balanced if every such interpretation as a ternary relation is balanced.

A relation that can be defined from the relations in Γ using only existential quantification, conjunction and equalities between variables is said to be *pp-definable.* Γ is *strongly balanced* if all pp-definable relations of arity three or more are balanced. This gives the criterion of the dichotomy in [10].

▶ **Theorem 1** (Dichotomy Theorem). *If Γ is strongly balanced, then #CSP(Γ) is in* FP*; otherwise, it is #*P*-complete.*

Note that infinitely many relations are pp-definable in Γ, which is why decidability is not obvious. Bulatov's criterion is equivalent but expressed in terms of an infinite algebra constructed from Γ so, again, is not obviously decidable.

In the remainder of the paper, we construct a nondeterministic, polynomial-time algorithm that determines whether a given constraint langauge Γ is strongly balanced.

1.1 Proof outline

Our proof of the Dichotomy Theorem [10] uses succinct representations, which we call "frames", of a class of relations we call *strongly rectangular.* We do not require frames in the present paper but strong rectangularity is useful as it imposes structure and because every strongly balanced relation is strongly rectangular. We first show that strong rectangularity is decidable in NP.

We next develop an alternative, equational characterisation of strong balance. We use this characterisation to translate the question of whether a constraint langauge Γ over domain D is strongly balanced to a property of homomorphisms to the relational structure $(D, \Gamma)^6$ (we use a standard definition of Cartesian products). Using a technique due to Lovász [16], we show that this property is equivalent to the existence of certain automorphisms of the product structure. It follows that strong balance is decidable in NP, since we can nondeterministically "guess" a suitable collection of functions and check, in deterministic polynomial time, that they are the desired automorphisms.

1.2 Organisation of the paper

The remainder of the paper is organised as follows. The necessary definitions and notation and some basic results appear in Section 2. In Section 3, we review the concept of strong rectangularity, which we introduced in [10] and, in Section 4, we formally define strong balance and present some necessary results on rank-one block matrices. The proof of the decidability of strong balance appears in Section 5 and some concluding remarks follow, in Section 6.

2 Definitions and notation

Given a set D, we write $\mathbf{a} = (a_1, \dots, a_r)$ for an r-ary tuple in D^r. We will sometimes omit the brackets and commas and just write $a_1 \dots a_r$.

For a natural number n, we write $[n]$ for the set $\{1, \dots, n\}$.

2.1 Relations and constraints

Let $D = \{d_1, d_2, \dots, d_q\}$ be a finite *domain* with $q = |D|$. A *constraint language* Γ is a finite set of named, finitary relations on D, including the binary equality relation $\{(d_i, d_i) : i \in [q]\}$, which we denote by $=$. We will call $\mathfrak{G} = (D, \Gamma)$ a *relational structure*. We may view an r-ary relation H on D with $\ell = |H|$ as an $\ell \times r$ matrix with elements in D. Then a tuple $\mathbf{t} \in H$ is any row of this matrix. We write H^Γ for the instantiation of the relation H in Γ.

We define the *size* of a relation H as $\|H\| = \ell r$, the number of elements in its matrix, and the size of Γ as $\|\Gamma\| = \sum_{H \in \Gamma} \|H\|$. To avoid trivialities, we will assume that every relation $H \in \Gamma$ is nonempty. We will also assume that every $d \in D$ appears in a tuple of some relation $H \in \Gamma$. If this is not so for some d, we can remove it from D. It then follows that $\|\Gamma\| \geq q$.

Let $V = \{\nu_1, \nu_2, \dots, \nu_n\}$ be a finite *codomain*. An *assignment* is a function $\mathbf{x} \colon V \to D$. We will abbreviate $\mathbf{x}(\nu_i)$ to x_i. If $\{i_1, i_2, \dots, i_r\} \subseteq [n]$, we write $H(x_{i_1}, x_{i_2}, \dots, x_{i_r})$ for the relation $\Theta = \{\mathbf{x} : (x_{i_1}, x_{i_2}, \dots, x_{i_r}) \in H\}$ and we refer to this as a *constraint*. Then $(\nu_{i_1}, \nu_{i_2}, \dots, \nu_{i_r})$ is the *scope* of the constraint and we say that $\mathbf{x}$ is a *satisfying* assignment for the constraint if $\mathbf{x} \in \Theta$.

A Γ-*formula* Φ in a set of variables $\{x_1, x_2, \dots, x_n\}$ is a conjunction of constraints $\Theta_1 \wedge \dots \wedge \Theta_m$. We will identify the variables with the x_i above, although strictly the latter are only a *model* of the formula. The precise labelling of the variables is of no significance and a formula remains the same if its variables are bijectively renamed.

A Γ-formula Φ describes an instance of the *constraint satisfaction problem* (CSP) with constraint language Γ. A satisfying assignment for Φ is an assignment that satisfies all Θ_i ($i \in [m]$). The set of all satisfying assignments for Φ is the Γ-*definable* relation R_Φ over D. We will make no distinction between Φ and R_Φ, unless this could cause confusion.

If $H \subseteq D^r$ and $I = \{i_1, \dots, i_k\} \subseteq [r]$, with $i_1 < \dots < i_k$, we write $\mathsf{pr}_I H$ for the *projection* of H given by $\{(a_{i_1}, \dots, a_{i_k}) : a_1 \dots a_r \in H\}$.

2.2 Definability

A *primitive positive* (pp) formula Ψ is a Γ-formula Φ with existential quantification over some subset of the variables. A satisfying assignment for Ψ is any satisfying assignment for Φ. The unquantified (free) variables then determine the *pp-definable* relation R_Ψ, a projection of R_Φ. Again, we make no distinction between Ψ and R_Ψ.

The set of all Γ-definable relations is denoted by $\mathsf{CSP}(\Gamma)$ and the set of all relations pp-definable in Γ is the *relational clone* $\langle\Gamma\rangle$.

2.3 Polymorphisms

A *Mal'tsev polymorphism* of a relation $H \subseteq D^r$ is a function $\varphi\colon D^3 \to D$ with the following properties:

1. whenever $\mathbf{a}, \mathbf{b}, \mathbf{c} \in H$, we have $\varphi(\mathbf{a}, \mathbf{b}, \mathbf{c}) := \big(\varphi(a_1, b_1, c_1), \ldots, \varphi(a_r, b_r, c_r)\big) \in H$;
2. for any $a, b \in D$, $\varphi(a, b, b) = \varphi(b, b, a) = a$.

The first condition describes a (ternary) polymorphism; the second is known as the Mal'tsev property. Note that the first condition can be extended to functions of arbitrary arity but we only require ternary polymorphisms here.

A function φ is a polymorphism of a constraint langauge Γ if it is a polymorphism of every relation in Γ. The following lemma is well known from the folklore and is easy to prove.

▶ **Lemma 2.** *φ is a polymorphism of Γ if, and only if, it is a polymorphism of $\langle\Gamma\rangle$.*

2.4 Homomorphisms and monomorphisms

A different, but equivalent, view of $\mathsf{CSP}(\Gamma)$ is often taken in the literature. This is to regard Φ as a finite structure with domain V and relations determined by the scopes of the constraints. Thus, we have relations $\tilde{H}$, where $(i_1, i_2, \ldots, i_r) \in \tilde{H}$ if, and only if, $H(x_{i_1}, x_{i_2}, \ldots, x_{i_r})$ is a constraint. A satisfying assignment $\mathbf{x}$ corresponds to a *homomorphism* from Φ to Γ.

The following definitions and notation are used only in Section 5. Let $[D_1 \to D_2]$ denote the set of functions from D_1 to D_2. A homomorphism between two relational structures $\mathfrak{S}_1 = (D_1, \Gamma_1)$ and $\mathfrak{S}_2 = (D_2, \Gamma_2)$ is a function $\sigma \in [D_1 \to D_2]$ that preserves relations. Thus, for each r-ary relation H and each tuple $\mathbf{u} = (u_1, \ldots, u_r) \in H^{\Gamma_1}$, we have $\sigma(\mathbf{u}) = (\sigma(u_1), \ldots, \sigma(u_r)) \in H^{\Gamma_2}$. We write $\sigma\colon \mathfrak{S}_1 \to \mathfrak{S}_2$ to indicate that σ is a homomorphism.

Let $[V \hookrightarrow D]$ and $[V \leftrightarrow D]$ denote the sets of all *injective* and *bijective* functions $V \to D$, respectively. An injective homomorphism is called a *monomorphism* and we will write $\sigma\colon \mathfrak{S}_1 \hookrightarrow \mathfrak{S}_2$. An *endomorphism* of a relational structure $\mathfrak{S}$ is a homomorphism $\sigma\colon \mathfrak{S} \to \mathfrak{S}$ (such a function is also a unary polymorphism). An *automorphism* is a bijective endomorphism whose inverse is also an endomorphism. Note that $[\mathfrak{S} \hookrightarrow \mathfrak{S}] = [\mathfrak{S} \leftrightarrow \mathfrak{S}]$, since D is finite, so an injective endomorphism is always an automorphism. Clearly, the identity function is always an automorphism, for any relational structure $\mathfrak{S}$.

2.5 Powers of structures

We use the following construction of *powers of* $\mathfrak{S}$ (see, for example, [17, p. 282]). For any relational structure $\mathfrak{S} = (D, \Gamma)$ and $k \in \mathbb{N}$, the relational structure $\mathfrak{S}^k = (D^k, \Gamma^k)$ is defined as follows. The domain is the Cartesian power D^k. The constraint language Γ^k is such that, each r-ary relation $H \in \Gamma$, corresponds to an r-ary $H^k \in \Gamma^k$, which is defined as follows. If $\mathbf{u}_i = (u_{i,1}, u_{i,2}, \ldots, u_{i,k}) \in D^k$ $(i \in [r])$, then $(\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_r) \in H^k$ if, and only if, $(u_{1,j}, u_{2,j}, \ldots, u_{r,j}) \in H$ for all $j \in [k]$. Now, if Ψ is a formula pp-definable in Γ, we define the corresponding formula Ψ^k to be identical to Ψ, except that each occurrence of $H \in \Gamma$ is replaced by the corresponding relation $H^k \in \Gamma^k$. Observe that Ψ^k is actually pp-definable in Γ, since $\Psi^k(\mathbf{x}) = \Psi(\mathbf{x}_1) \wedge \Psi(\mathbf{x}_2) \wedge \cdots \wedge \Psi(\mathbf{x}_k)$, where the $\mathbf{x}_i$ $(i \in [k])$ are disjoint n-tuples of variables. In particular, we have $|\Psi^k| = |\Psi|^k$.

Using this construction, the definition of a polymorphism can be reformulated. In this view of $\mathsf{CSP}(\Gamma)$, it follows directly that a k-ary polymorphism is just a homomorphism $\varphi\colon \mathfrak{S}^k \to \mathfrak{S}$.

3 Rectangularity

The key tool in our proof of the Dichotomy Theorem [10] is the use of succint representations (which we call "frames") for a class of relations that we call strongly rectangular. Frames allow us to efficiently count solutions in the polynomial-time cases but we do not require them here. However, the concept of strong rectangularity does play a role in our analysis.

A binary relation $H \subseteq A_1 \times A_2$ is *rectangular* if, for all $a, b \in A_1$ and $c, d \in A_2$,

$$ac, ad, bc \in H \text{ implies } bd \in H\,.$$

For $r \geq 2$, a relation $H \subseteq D^r$ can be considered as a binary relation in $D^k \times D^{r-k}$ for any k with $1 \leq k < r$. We say that a relation of arity $r \geq 2$ is rectangular if every such expression of it as a binary relation is rectangular.

▶ **Definition 3.** A constraint langauge Γ is strongly rectangular if every relation in $\langle\Gamma\rangle$ of arity ≥ 2 is rectangular.

We consider the following computational problem.

STRONG RECTANGULARITY

Instance : A relational structure $\mathfrak{G} = (D, \Gamma)$.

Question : Is Γ strongly rectangular?

As $\langle\Gamma\rangle$ is an infinite set, it is not immediate whether STRONG RECTANGULARITY is decidable. However, it turns out that strong rectangularity is equivalent to the existence of a Mal'tsev polymorphism.

▶ **Lemma 4.** *Γ is strongly rectangular if, and only if, it has a Mal'tsev polymorphism.*

We defer the proof of this lemma for a moment. We require the lemma to prove the following result.

▶ **Lemma 5.** STRONG RECTANGULARITY *is in* NP.

Proof. By Lemma 4, Γ is strongly rectangular if, and only if, it has a Mal'tsev polymorphism. Thus, we nondeterministically guess a function $\varphi\colon D^3 \to D$ in time $O(q^3)$. We can verify that φ is a Mal'tsev polymorphism, deterministically in time $O(\|\Gamma\|^4)$ just by checking that all relevant inputs to φ produce appropriate outputs. ◀

Lemma 4 is usually proved in an algebraic setting. That proof is not difficult, but requires an understanding of concepts from universal algebra, such *free algebras* and *varieties* [6]. Therefore, we will give a proof in the relational setting which, we believe, provides more insight for the reader whose primary interest is in relations.

Proof of Lemma 4. Suppose Γ has a Mal'tsev polymorphism φ. Consider any pp-definable binary relation $B \subseteq D^r \times D^s$. By Lemma 2, φ is also a polymorphism of B. If $(\mathbf{a}, \mathbf{c})$, $(\mathbf{a}, \mathbf{d})$, $(\mathbf{b}, \mathbf{d}) \in B$ then we have $(\varphi(\mathbf{a}, \mathbf{a}, \mathbf{b}), \varphi(\mathbf{c}, \mathbf{d}, \mathbf{d})) = (\mathbf{b}, \mathbf{c}) \in B$, from the definition of a Mal'tsev polymorphism. Thus, B is rectangular and, hence, Γ is strongly rectangular.

Conversely, suppose Γ is strongly rectangular. Denote the relation $H \in \Gamma$ by $H = \{\mathbf{u}_i^H : i \in [\ell_H]\}$, where $\mathbf{u}_i^H \in D^{r_H}$. Consider the Γ-formula

$$\Phi(\mathbf{x}) \;=\; \bigwedge_{H\in\Gamma}\ \bigwedge_{i_1\in[\ell_H]}\ \bigwedge_{i_2\in[\ell_H]}\ \bigwedge_{i_3\in[\ell_H]} H\big(\mathbf{x}^H_{i_1,i_2,i_3}\big)\,,$$

where $\mathbf{x}^H_{i_1,i_2,i_3}$ is an r_H-tuple of variables, distinct for all $H \in \Gamma$, $i_1, i_2, i_3 \in [\ell_H]$. Thus, the relation R_Φ has arity $r_\Phi = \sum_{H\in\Gamma} r_H \ell_H^3$ and $|R_\Phi| = \prod_{H\in\Gamma} {\ell_H}^{\ell_H^3}$.

Clearly R_Φ has three tuples $\mathbf{u}_1$, $\mathbf{u}_2$, $\mathbf{u}_3$ such that the sub-tuple corresponding to $\mathbf{x}^H_{i_1,i_2,i_3}$ in $\mathbf{u}_j$ is $\mathbf{u}^H_{i_j}$ for each $j \in \{1, 2, 3\}$. Then $U = \{\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3\}$ has the following universality property for Γ. For all $H \in \Gamma$ and every triple of (not necessarily distinct) tuples $\mathbf{t}_1$, $\mathbf{t}_2$, $\mathbf{t}_3 \in H$, there is a set $I(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3)$ with $I \subseteq [r_\Phi]$, $|I| = r_H$ such that $\mathsf{pr}_I R_\Phi = H$ and $\mathsf{pr}_I \mathbf{u}_j = \mathbf{t}_j$ $(j = 1, 2, 3)$.

Now, for each set of identical columns in U, we impose equality on the corresponding variables in Φ, to give a Γ-formula Φ'. Let U' be the resulting submatrix of U, with rows $\mathbf{u}'_1$, $\mathbf{u}'_2$, $\mathbf{u}'_3$. Observe that U' is obtained by deleting copies of columns in U. Therefore U' has no identical columns and has a column (a, b, c) for all $a, b, c \in \mathsf{pr}_k H$ with $H \in \Gamma$ and $k \in [r_H]$.

Next, for all columns (a, b, c) of U' such that $b \notin \{a, c\}$, we impose existential quantification on the corresponding variables in Φ', to give a pp-formula Φ''. Let U'' be the submatrix of U' with rows $\mathbf{u}''_1$, $\mathbf{u}''_2$, $\mathbf{u}''_3$ corresponding to $\mathbf{u}'_1$, $\mathbf{u}'_2$, $\mathbf{u}'_3$. Then U'' results from deleting columns in U' and U'' has columns of the form (a, a, b) or (c, d, d). Thus, after rearranging columns (relabelling variables), we will have

$$U'' = \begin{bmatrix} \mathbf{u}''_1 \\ \mathbf{u}''_2 \\ \mathbf{u}''_3 \end{bmatrix} = \begin{bmatrix} \mathbf{a} & \mathbf{c} \\ \mathbf{a} & \mathbf{d} \\ \mathbf{b} & \mathbf{d} \end{bmatrix},$$

for some nonempty tuples $\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{d}$. By strong rectangularity, this implies $\mathbf{u}'' = \begin{bmatrix} \mathbf{b} & \mathbf{c} \end{bmatrix} \in R_{\Phi''}$.

Removing the existential quantification in Φ'', $\mathbf{u}''$ can be extended to $\mathbf{u}' \in R_{\Phi'}$. Now, if column k of U' is (a, b, c) say, we define $\varphi(a, b, c) = u'_k$. This is unambiguous, since U' has no identical columns. Thus, $\mathbf{u}' = \varphi(\mathbf{u}'_1, \mathbf{u}'_2, \mathbf{u}'_3) \in R_{\Phi'}$. If, for any $a, b, c \in D$, $\varphi(a, b, c)$ remains undefined, we will set $\varphi(a, b, c) = a$ unless $a = b$, in which case $\varphi(a, b, c) = c$. Clearly φ satisfies $\varphi(a, b, b) = \varphi(b, b, a) = a$, for all $a, b \in D$, and so has the Mal'tsev property.

Removing the equalities between variables in Φ', $\mathbf{u}'$ can be further extended to $\mathbf{u} = \varphi(\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3) \in R_\Phi$. This is consistent since $\mathbf{u}$ satisfies the equalities imposed on Φ to give Φ'. Now, for any $\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3 \in H$, the universality property of U implies that $\mathsf{pr}_I \mathbf{u} = \varphi(\mathbf{t}_1, \mathbf{t}_2, \mathbf{t}_3) \in H$. Thus, φ preserves all $H \in \Gamma$, so it is a polymorphism and hence a Mal'tsev polymorphism. ◀

4 Strong balance

Recall the Dichotomy Theorem (Theorem 1): $\#\mathsf{CSP}(\Gamma)$ is computable in polynomial time if Γ is strongly balanced, and is $\#\mathsf{P}$-complete, otherwise. In this section, we formally define strong balance and investigate its properties.

A *rank-one block matrix* is a $k \times k$ matrix M whose rows and columns can be permuted to give a block-diagonal matrix whose non-zero blocks have rank 1. (It is equivalent to say that the relation $\{xy : M(x, y) \neq 0\}$ is rectangular and there are functions $\alpha, \beta \colon [k] \to \mathbb{N}$ such that $M(x, y) = \alpha(x)\beta(y)$ where $M \neq 0$ but we will use a third characterisation, given by Corollary 9.)

Let $H \subseteq A_1 \times A_2 \times A_3$ be a ternary relation. We say that H is *balanced* if the *balance matrix*

$$M(x, y) = |\{z \in A_3 : (x, y, z) \in H\}| \qquad (x \in A_1,\ y \in A_2)$$

is a rank-one block matrix. For $r > 3$, a relation $H \subseteq D^r$ can be expressed as a ternary relation in $D^k \times D^\ell \times D^{r-k-\ell}$ for any $k, \ell \geq 1$ with $k + \ell < r$. We say that a relation H of arity $r > 3$ is balanced if every such expression of H as a ternary relation is balanced.

▶ **Definition 6.** A constraint language Γ is *strongly balanced* if every relation of arity ≥ 3 in $\langle\Gamma\rangle$ is balanced.

Note that infinitely many relations are pp-definable in any constraint language. Our goal in the remainder of this paper is to show that, this notwithstanding, the property of being strongly balanced is decidable.

Towards this goal, we derive a different characterisation of rank-one block matrices. This may seem more complicated than the original definition, but it is more suited to our purpose. By the *underlying relation* of a matrix $M(x, y)$, we mean the relation $\{(x, y) : M(x, y) \neq 0\}$. We say that a matrix is rectangular if its underlying relation is.

▶ **Lemma 7.** *A is a rank-one block matrix if, and only if, every 2×2 submatrix of A is a rank-one block matrix.*

Proof. Let A be a $k \times \ell$ rank-one block matrix and let

$$B = \begin{bmatrix} a_{ir} & a_{is} \\ a_{jr} & a_{js} \end{bmatrix} \qquad (i, j \in [k],\ r, s \in [\ell])$$

be any 2×2 submatrix of A. If any of a_{ir}, a_{is}, a_{jr}, a_{js} is zero, at least two must be zero, since A is rectangular. In this case, B is clearly a rank-one block matrix. If a_{ir}, a_{is}, a_{jr}, a_{js} are all nonzero, B must be a submatrix of some block of A. Since this block has rank one, B also has rank one.

Conversely, suppose A is not a rank-one block matrix. If its underlying relation is not rectangular, there exist $a_{ir}, a_{is}, a_{jr} > 0$ with $a_{js} = 0$. The corresponding matrix B clearly has rank two, and has only one block so is not a rank-one block matrix. If the underlying relation of A is rectangular, then A must have a block of rank at least two. This block must have some 2×2 submatrix B with rank two and all its elements non-zero. ◀

▶ **Lemma 8.** *A rectangular 2×2 matrix A is a rank-one block matrix if, and only if, $a_{11}^2 a_{22}^2 a_{12} a_{21} = a_{12}^2 a_{21}^2 a_{11} a_{22}$.*

Proof. The equation holds if any of a_{11}, a_{12}, a_{21} or a_{22} is zero. But, then, rectangularity implies that at least two of them must be zero and A is a rank-one block matrix in all possible cases. Otherwise, the equation is equivalent to $a_{11}a_{22} = a_{12}a_{21}$, which is the condition that A is singular. So A is one block, with rank one. The argument is clearly reversible. ◀

▶ **Corollary 9.** *A rectangular $k \times \ell$ matrix A is a rank-one block matrix if, and only if, $a_{ir}^2 a_{js}^2 a_{is} a_{jr} = a_{is}^2 a_{jr}^2 a_{ir} a_{js}$, for all $i, j \in [k]$ and $r, s \in [\ell]$.*

Proof. When $i = j$ or $r = s$, the two sides of the equation are identical. Otherwise, the equality follows directly from Lemmas 7 and 8. ◀

It is possible to modify the above so that Corollary 9 involves products of only five elements, rather than six, but we do not pursue that refinement here.

The following lemma gives a basic precondition for strong balance.

▶ **Lemma 10.** *Every strongly balanced constraint language is strongly rectangular.*

Proof. Suppose $\Psi \in \langle \Gamma \rangle$ is not rectangular. There are $\mathbf{ac}, \mathbf{bc}, \mathbf{ad} \in R$ such that $\mathbf{bd} \notin R$. Let Ψ' be the relation $\{\mathbf{uvv} : \mathbf{uv} \in \Psi\}$ and let M' be its balance matrix. The 2×2 submatrix corresponding to rows $\mathbf{a}$ and $\mathbf{b}$ and columns $\mathbf{c}$ and $\mathbf{d}$ is

$$B = \begin{bmatrix} \alpha & \beta \\ \gamma & 0 \end{bmatrix}$$

for some $\alpha, \beta, \gamma \geq 1$. This submatrix is not a rank-one block matrix so, by Lemma 7, nor is M'. Therefore, Γ is not strongly balanced. ◀

5 Decidability

We now give a relaxation of the strong balance criterion, by noting the conditions sufficient for the success of the algorithm in [10]. For an instance with n variables, the algorithm only requires that ternary relations on $D \times D \times D^i$, for $i \in [n-2]$, be balanced. Therefore, let $\Psi(\mathbf{x})$, with $\mathbf{x} = (x_1, \ldots, x_n)$, be an arbitrary formula pp-definable in Γ. For the algorithm to succeed, it suffices that the $q \times q$ matrix

$$M(a,b) = \big|\{\mathbf{x} \in [V \to D] : \mathbf{x} \in \Psi,\, x_1 = a,\, x_2 = b\}\big| \qquad (\forall a, b \in D)$$

is a rank-one block matrix for any Ψ. We may therefore take this as the criterion for strong balance.

We will construct an algorithm to solve the following decision problem.

STRONG BALANCE

Instance : A relational structure $\mathfrak{S} = (D, \Gamma)$.

Question : Is Γ strongly balanced?

Recall from Section 2 that we may assume that $\|\Gamma\| \geq q$. Thus, we may take $\|\Gamma\|$ as the measure of input size for STRONG BALANCE and we bound the complexity of STRONG BALANCE as a function of this value. Complexity is a secondary issue, since $\|\Gamma\|$ is a constant in the nonuniform model for #CSP(Γ). In this model, we are only required to show that some algorithm exists to solve STRONG BALANCE. However, we believe that the computational complexity of deciding the dichotomy is intrinsically interesting. Our approach will be to show that the strong balance condition is equivalent to a structural property of Γ that can be checked in NP.

We may assume that Γ is strongly rectangular since, if it is not, we know by Lemma 10 that it is not strongly balanced. For the remainder of this section, we fix an n-ary pp-definable relation $\Psi \in \langle \Gamma \rangle$ with balance matrix M.

By Corollary 9, the condition for M to be a rank-one block matrix is that

$$M(a,c)^2 M(a,d) M(b,d)^2 M(b,c) = M(a,d)^2 M(a,c) M(b,c)^2 M(b,d) \quad \text{for all } a,b,c,d \in D.$$

We can reformulate this condition using the construction of powers of $\mathfrak{S}$. If $\mathbf{a} = (a_1, \ldots, a_k)$ and $\mathbf{b} = (b_1, \ldots, b_k)$, the balance matrix M_k for Ψ^k is the $q^k \times q^k$ matrix given by

$$\begin{aligned} M_k(\mathbf{a}, \mathbf{b}) &= \big|\{\mathbf{x} \in [V \to D^k] \cap \Psi^k : x_1 = \mathbf{a},\, x_2 = \mathbf{b}\}\big| \\ &= M(a_1, b_1) M(a_2, b_2) \cdots M(a_k, b_k)\,. \end{aligned}$$

The condition for M to be a rank-one block matrix can be rewritten as

$$M_6(\bar{a}, \bar{c}) = M_6(\bar{a}, \bar{d})\,, \tag{1}$$

where

$$\bar{a} = (a,a,a,b,b,b),\ \bar{c} = (c,c,d,d,d,c),\ \bar{d} = (d,d,c,c,c,d)\,. \tag{2}$$

Let us fix $\bar{a}$, $\bar{c}$, $\bar{d}$. For notational simplicity, let us write $\bar{\mathfrak{S}}$ for $\mathfrak{S}^6$, $\bar{\Gamma}$ for Γ^6, $\bar{\Psi}$ for Ψ^6, $\bar{M}$ for M_6 and $\bar{D}$ for D^6. Then, from (1), we must verify that $\bar{M}(\bar{a},\bar{c}) = \bar{M}(\bar{a},\bar{d})$ for all relations $R_{\bar{\Psi}}$ that are pp-definable in $\bar{\Gamma}$ and given $\bar{a},\bar{c},\bar{d} \in \bar{D}$. We use a method of Lovász [16]; see also [8]. For $\bar{s} \in \bar{D}$, and a pp-definition $\bar{\Psi}$ in variables V, let

$$\begin{aligned}\mathrm{Hom}_{\bar{s}}(\bar{\Psi}) &= \{\mathbf{x} \in [V \to \bar{D}] \cap \bar{\Psi} : x_1 = \bar{a},\ x_2 = \bar{s}\}\\ \mathrm{hom}_{\bar{s}}(\bar{\Psi}) &= |\mathrm{Hom}_{\bar{s}}(\bar{\Psi})|\,.\end{aligned}$$

However, a homomorphism $V \to \bar{D}$ that is consistent with $\bar{\Psi}$ is just a satisfying assignment to $\bar{\Psi}$. $\bar{M}(\bar{a},\bar{s})$ is the number of such assignments with $x_1 = \bar{a}$ and $x_2 = \bar{s}$, i.e., the number of homomorphisms that map $x_1 \mapsto \bar{a}$ and $x_2 \mapsto \bar{s}$. This proves the following.

▶ **Lemma 11.** Γ *is strongly balanced if, and only if,* $\mathrm{hom}_{\bar{c}}(\bar{\Psi}) = \mathrm{hom}_{\bar{d}}(\bar{\Psi})$ *for all formulae* $\bar{\Psi}$ *and all* $\bar{a},\bar{c},\bar{d}$ *of the form above.*

We will also need to consider the injective functions in $\mathrm{Hom}_{\bar{s}}(\bar{\Psi})$. For $\bar{s} \in \bar{D}$, let

$$\begin{aligned}\mathrm{Mon}_{\bar{s}}(\bar{\Psi}) &= \{\mathbf{x} \in [V \hookrightarrow \bar{D}] \cap \bar{\Psi} : x_1 = \bar{a},\ x_2 = \bar{s}\}\\ \mathrm{mon}_{\bar{s}}(\bar{\Psi}) &= |\mathrm{Mon}_{\bar{s}}(\bar{\Psi})|\,.\end{aligned}$$

▶ **Lemma 12.** $\mathrm{hom}_{\bar{c}}(\bar{\Psi}) = \mathrm{hom}_{\bar{d}}(\bar{\Psi})$ *for all* $\bar{\Psi}$ *if, and only if,* $\mathrm{mon}_{\bar{c}}(\bar{\Psi}) = \mathrm{mon}_{\bar{d}}(\bar{\Psi})$ *for all* $\bar{\Psi}$.

Proof. Consider the set $\mathcal{I}$ of all partitions I of V into disjoint classes $\bar{I}_1,\dots,\bar{I}_{k_I}$, such that $1 \in \bar{I}_1$, $2 \in \bar{I}_2$. Writing $I \preceq I'$ whenever I is a refinement of I', $\mathbb{P} = (\mathcal{I},\preceq)$ is a poset. We will write $\perp$ for the partition into singletons, so $\perp \preceq I$ for all $I \in \mathcal{I}$.

Let V/I denote the set of classes $\bar{I}_1,\dots,\bar{I}_{k_I}$ of the partition I, so $|V/I| = k_I$, and let $\bar{I}_1$, $\bar{I}_2$ be denoted by $1/I$, $2/I$. Let $\bar{\Psi}/I$ denote the relation obtained from $\bar{\Psi}$ by imposing equality on all pairs of variables that occur in the same partition of I. Thus, the constraints $x_1 = \bar{a}$, $x_2 = \bar{s}$ become $x_{1/I} = \bar{a}$, $x_{2/I} = \bar{s}$. Then we have

$$\mathrm{hom}_{\bar{s}}(\bar{\Psi}) \;=\; \mathrm{hom}_{\bar{s}}(\bar{\Psi}/\perp) \;=\; \sum_{I\in\mathcal{I}} \mathrm{mon}_{\bar{s}}(\bar{\Psi}/I) \;=\; \sum_{I\in\mathcal{I}} \mathrm{mon}_{\bar{s}}(\bar{\Psi}/I)\zeta(\perp,I)\,, \tag{3}$$

where $\zeta(I,I') = 1$, if $I \preceq I'$, and $\zeta(I,I') = 0$, otherwise, is the ζ-function of $\mathbb{P}$. Thus, if $\mathrm{mon}_{\bar{c}}(\bar{\Psi}) = \mathrm{mon}_{\bar{d}}(\bar{\Psi})$ for all $\bar{\Psi}$, then

$$\mathrm{hom}_{\bar{c}}(\bar{\Psi}) \;=\; \sum_{I\in\mathcal{I}} \mathrm{mon}_{\bar{c}}(\bar{\Psi}/I)\zeta(\perp,I) \;=\; \sum_{I\in\mathcal{I}} \mathrm{mon}_{\bar{d}}(\bar{\Psi}/I)\zeta(\perp,I) \;=\; \mathrm{hom}_{\bar{d}}(\bar{\Psi})\,. \tag{4}$$

Conversely, suppose that $\mathrm{hom}_{\bar{c}}(\bar{\Psi}) = \mathrm{hom}_{\bar{d}}(\bar{\Psi})$ for all $\bar{\Psi}$. The reasoning used to give (3) implies, more generally, that

$$\mathrm{hom}_{\bar{s}}(\bar{\Psi}/I) \;=\; \sum_{I\preceq I'} \mathrm{mon}_{\bar{s}}(\bar{\Psi}/I') \;=\; \sum_{I'\in\mathcal{I}} \mathrm{mon}_{\bar{s}}(\bar{\Psi}/I')\zeta(I,I')\,.$$

Now, Möbius inversion for posets [20, Ch. 25] implies that the matrix $\zeta\colon \mathcal{I}\times\mathcal{I} \to \{0,1\}$ has an inverse $\mu\colon \mathcal{I}\times\mathcal{I} \to \mathbb{Z}$. It follows directly that

$$\mathrm{mon}_{\bar{s}}(\bar{\Psi}) \;=\; \sum_{I\in\mathcal{I}} \mathrm{hom}_{\bar{s}}(\bar{\Psi}/I)\mu(\perp,I)\,.$$

Thus, with $\hom_{\bar{c}}(\bar{\Psi}) = \hom_{\bar{d}}(\bar{\Psi})$ for all $\bar{\Psi}$, we have

$$\mathrm{mon}_{\bar{c}}(\bar{\Psi}) \;=\; \sum_{I \in \mathcal{I}} \hom_{\bar{c}}(\bar{\Psi}/I)\mu(\perp, I) \;=\; \sum_{I \in \mathcal{I}} \hom_{\bar{d}}(\bar{\Psi}/I)\mu(\perp, I) \;=\; \mathrm{mon}_{\bar{d}}(\bar{\Psi})\,. \tag{5}$$

Now, (4) and (5) give the conclusion. ◀

▶ **Lemma 13.** $\mathrm{mon}_{\bar{c}}(\bar{\Psi}) = \mathrm{mon}_{\bar{d}}(\bar{\Psi})$ *for all* $\bar{\Psi}$*, if, and only if, there is an automorphism* $\eta\colon \bar{D} \leftrightarrow \bar{D}$ *of* $\bar{\mathfrak{S}} = (\bar{D}, \bar{\Gamma})$ *such that* $\eta(\bar{a}) = \bar{a}$ *and* $\eta(\bar{c}) = \bar{d}$*.*

Proof. The condition holds if $\bar{\mathfrak{S}}$ has such an automorphism since, if $\bar{\Psi}(\mathbf{x}) = \exists \mathbf{y}\, \bar{\Phi}(\mathbf{x}, \mathbf{y})$ for some $\bar{\Phi}$, then

$$\begin{aligned}
\mathrm{mon}_{\bar{c}}(\bar{\Psi}) \;&=\; |\{\mathbf{x} \in [V \hookrightarrow \bar{D}] : x_1 = \bar{a},\, x_2 = \bar{c},\, \exists \mathbf{y}\, (\mathbf{x}, \mathbf{y}) \in \bar{\Phi}\}| \\
&=\; |\{\eta(\mathbf{x}) \in [V \hookrightarrow \bar{D}] : x_1 = \eta(\bar{a}),\, x_2 = \eta(\bar{c}),\, \exists \mathbf{y}\, (\eta(\mathbf{x}), \eta(\mathbf{y})) \in \bar{\Phi}\}| \\
&=\; |\{\mathbf{x} \in [V \hookrightarrow \bar{D}] : x_1 = \bar{a},\, x_2 = \bar{d},\, \exists \mathbf{y}\, (\mathbf{x}, \mathbf{y}) \in \bar{\Phi}\}| \\
&=\; \mathrm{mon}_{\bar{d}}(\bar{\Psi})\,.
\end{aligned}$$

Conversely, suppose we have $\mathrm{mon}_{\bar{c}}(\bar{\Psi}) = \mathrm{mon}_{\bar{d}}(\bar{\Psi})$ for all $\bar{\Psi}$. Consider the following $\bar{\Gamma}$-formula $\bar{\Phi}$ with domain $\bar{D}$ and variables $x_{\bar{t}}$ ($\bar{t} \in \bar{D}$):

$$\bar{\Phi}(\mathbf{x}) = \bigwedge_{\bar{H} \in \bar{\Gamma}} \; \bigwedge_{(\bar{u}_1, \ldots, \bar{u}_r) \in \bar{H}} \bar{H}(x_{\bar{u}_1}, \ldots, x_{\bar{u}_r})\,.$$

Then

$$\mathrm{Mon}_{\bar{s}}(\bar{\Phi}) = \{\mathbf{x} \in [\bar{D} \hookrightarrow \bar{D}] : x_{\bar{a}} = \bar{a},\, x_{\bar{c}} = \bar{s},\, \mathbf{x} \in \bar{\Phi}\}\,.$$

We have $\mathrm{Mon}_{\bar{c}}(\bar{\Phi}) \neq \emptyset$, since the identity assignment $x_{\bar{t}} = \bar{t}$ for all $\bar{t} \in \bar{D}$ is clearly satisfying. Thus, by the assumption, $\mathrm{Mon}_{\bar{d}}(\bar{\Phi}) \neq \emptyset$. Let $\eta \in \mathrm{Mon}_{\bar{d}}(\bar{\Phi})$, so η is an endomorphism of $\bar{\mathfrak{S}}$ with $\eta(\bar{a}) = \bar{a}$ and $\eta(\bar{c}) = \bar{d}$. Since $[D \hookrightarrow D] = [D \leftrightarrow D]$, $\eta\colon D \leftrightarrow D$ is the required automorphism. ◀

▶ **Corollary 14.** $\mathfrak{S} = (D, \Gamma)$ *is strongly balanced if, and only if, for all* $\bar{a}, \bar{c}, \bar{d}$ *as defined in* (2)*,* $\bar{\mathfrak{S}} = (\bar{D}, \bar{\Gamma})$ *has an automorphism* ψ *such that* $\psi(\bar{a}) = \bar{a}$ *and* $\psi(\bar{c}) = \bar{d}$*.*

Proof. This follows from Lemmas 11, 12 and 13. ◀

This characterisation of strong balance leads directly to a nondeterministic algorithm.

▶ **Theorem 15.** STRONG BALANCE *is in* NP*.*

Proof. We first determine whether Γ is strongly rectangular, using the method of Lemma 5. If it is not, then Γ is not strongly balanced, by Lemma 10.

Otherwise, we can construct $\bar{\mathfrak{S}} = (\bar{D}, \bar{\Gamma})$ in time $O(\|\Gamma\|^6)$. Let $\bar{q} = q^6 = |\bar{D}|$ and let Π denote the set of $\bar{q}\,!$ permutations of $\bar{D}$. Each $\pi \in \Pi$ is a function $\pi\colon \bar{D} \hookrightarrow \bar{D}$ and so a potential automorphism of $\bar{\mathfrak{S}}$. For each of the q^4 possible choices $a, b, c, d \in D$, we determine $\bar{a}, \bar{c}, \bar{d} \in \bar{D}$ in polynomial time. We select $\pi \in \Pi$ nondeterministically and check that $\pi(\bar{a}) = \bar{a}$, $\pi(\bar{c}) = \bar{d}$ and that π preserves all $\bar{H} \in \bar{\Gamma}$. The computation requires $O(q^4\|\bar{\Gamma}\|^2) = O(\|\Gamma\|^{16})$ time in total, so everything other than the $O(q^{10}) = O(\|\Gamma\|^{10})$ nondeterministic choices can be done deterministically in a polynomial number of steps. ◀

We have paid little attention to the efficiency of the computations in Theorem 15. If the elements of D are encoded as binary numbers in $[q]$, comparisons and nondeterministic choices require $O(\log q)$ bit operations, rather than the $O(1)$ operations in our accounting. On the other hand, membership in H^6 can be tested in $O(\|H\|)$ comparisons, rather than the $O(\|H\|^6)$ that we have allowed. This might be reduced further by storing H in a suitable data structure, instead of a simple matrix. As we have noted, Corollary 9 can be refined to products of five terms, which can be used to improve the algorithm of Theorem 15.

If we consider the domain D as fixed, the problem of deciding whether constraint languages over that domain D are strongly balanced is in deterministic polynomial time. With D fixed, there are a constant number of potential Mal'tsev polymorphisms that must be checked to determine strong rectangularity, and the numbers of tuples $\bar{a}$, $\bar{c}$, $\bar{d}$ and possible automorphisms on D are also fixed constants.

6 Conclusions

We have shown that there is an algorithm that determines whether a constraint language is strongly balanced. This means that the complexity dichotomy for $\#\mathsf{CSP}(\Gamma)$ is effective, thus answering the major open problem that arose from the proofs that the dichotomy exists [3, 10].

Although we have shown strong balance to be decidable in NP, we have only established an upper bound. We believe the complexity of the problem to be interesting in its own right. It is not hard to see that the problem of determining whether the automorphisms of Corollary 14 exist is reducible to the graph isomorphism problem. It therefore seems unlikely that strong balance is NP-complete as this would imply NP-completeness of graph isomorphism which would, in turn, imply the collapse of the polynomial hierarchy [19]. However, we leave open the question of whether strong balance is equivalent to graph isomorphism or whether more efficient algorithms exist.

Bulatov's proof of the $\#\mathsf{CSP}(\Gamma)$ dichotomy is expressed not in terms of strong balance but in terms of the "congruence singularity" of Γ (or, more precisely, of an algebra defined from Γ). We have shown the two conditions to be equivalent but it remains open if there is a direct proof that the property of congruence singularity is decidable.

References

1 A. A. Bulatov. A dichotomy theorem for constraint satisfaction problems on a 3-element domain. *Journal of the ACM*, 53(1):66–120, 2006.

2 A. A. Bulatov. The complexity of the counting constraint satisfaction problem. *Electronic Colloquium on Computational Complexity*, 14(093), 2007. (Revised Feb. 2009).

3 A. A. Bulatov. The complexity of the counting constraint satisfaction problem. In *Proc. 35th International Colloquium on Automata, Languages and Programming (Part 1)*, LNCS 5125, pp. 646–661. Springer, 2008.

4 J.-Y. Cai, P. Lu, and M. Xia. Holant problems and counting CSP. In *Proc. 41st Annual ACM Symposium on Theory of Computing (STOC 2009)*, pp. 715–724. ACM, 2009.

5 N. Creignou and M. Hermann. Complexity of generalized satisfiability counting problems. *Information and Computation*, 125(1):1–12, 1996.

6 K. Denecke and S. L. Wismath. *Universal Algebra and Applications in Theoretical Computer Science*. Chapman and Hall/CRC, 2002.

7 M. E. Dyer, L. A. Goldberg, and M. R. Jerrum. A complexity dichotomy for hypergraph partition functions. *Computational Complexity*, 19(4):605–633, 2010.

8 M. E. Dyer, L. A. Goldberg, and M. S. Paterson. On counting homomorphisms to directed acyclic graphs. *Journal of the ACM*, 54(6), 2007.

9 M. E. Dyer and C. S. Greenhill. The complexity of counting graph homomorphisms. *Random Structures and Algorithms*, 17(3–4):260–289, 2000. (Corrigendum in *Random Structures and Algorithms*, 25(3):346–352, 2004.)

10 M. E. Dyer and D. M. Richerby. On the complexity of #CSP. In *Proc. 42nd ACM Symposium on Theory of Computing (STOC 2010)*, pp. 725–734. ACM, 2010.

11 R. Fagin. Monadic generalized spectra. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 21:89–96, 1975.

12 R. Fagin, L. J. Stockmeyer, and M. Y. Vardi. On monadic NP vs monadic co-NP. *Information and Computation*, 120(1):78–92, 1995.

13 T. Feder and M. 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.

14 P. Hell and J. Nešetřil. On the complexity of H-coloring. *Journal of Combinatorial Theory (Series B)*, 48(1):92–110, 1990.

15 R. E. Ladner. On the structure of polynomial time reducibility. *Journal of the ACM*, 22(1):155–171, 1975.

16 L. Lovász. Operations with structures. *Acta. Math. Acad. Sci. Hung.*, 18:321–328, 1967.

17 J. Nešetřil, M. H. Siggers and L. Zádori. A combinatorial constraint satisfaction problem dichotomy classification conjecture. *European Journal of Combinatorics*, 31(1):280–296, 2010.

18 T. Schaefer. The complexity of satisfiability problems. In *Proc. 10th Annual ACM Symposium on Theory of Computing*, pp. 216–226. ACM Press, 1978.

19 U. Schöning. Graph isomorphism is in the low hierarchy. *Journal of Computer and System Sciences*, 37(3):312–323, 1988.

20 J. van Lint and R. Wilson. *A Course in Combinatorics.* (2nd ed.). CUP, 2001.

A speed-up of oblivious multi-head finite automata by cellular automata

Alex Borello[1], Gaétan Richard[2], and Véronique Terrier[2]

1 LIF (Laboratoire d'Informatique Fondamentale)
39, rue Frédéric-Joliot-Curie, 13453 Marseille, France
alex.borello@lif.univ-mrs.fr

2 GREYC (Groupe de Recherche en Informatique, Image, Automatique et Instrumentation de Caen)
Campus Côte-de-Nacre, Boulevard du Maréchal-Juin, 14032 Caen, France
gaetan.richard@info.unicaen.fr
veronique.terrier@info.unicaen.fr

Abstract

In this paper, we present a parallel speed-up of a simple, yet significantly powerful, sequential model by cellular automata. The simulated model is called oblivious multi-head finite automata and is characterized by the fact that the trajectory of the heads only depends on the length of the input word. While the original k-head finite automaton works in time $O(n^k)$, its corresponding cellular automaton performs the same task in time $O(n^{k-1}\log(n))$ and space $O(n^{k-1})$.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.2 Modes of Computation

Keywords and phrases oblivious multi-head finite automata, cellular automata, parallel speed-up, simulation

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.273

Introduction

Cellular automata (CA for short) are recognized as a major model of massively parallel computation. Their simple and homogeneous description as well as their ability to distribute and synchronize the information in a very efficient way contribute to their success. However, to determine to what extent CA can fasten sequential computation is not a simple task.

As regards specific sequential problems, the gain in speed by the use of CA is manifest [1, 2, 3]. But when we try to get general simulations, we have to face the delicate question of whether parallel algorithms are always faster than sequential ones. An inherent difficulty arises from the fact that efficient parallel algorithms make often use of techniques radically different from the sequential ones. Also there might exist a faster CA for each singular sequential solution whereas no general simulation exists.

Hence, no surprise: for Turing machines, model of sequential computation, the known simulations by CA provide no parallel speed-up. The early construction of Smith [9] simulates one step of the Turing machine by one step of the CA. Furthermore, no faster simulations have been reported yet even for restricted variants. In particular, we do not know whether any finite automata with k heads can be simulated on CA in less than $O(n^k)$ steps, which is the sequential time complexity.

In a step toward addressing such issues, we shall examine here a simple sequential model, called data-independent multi-head finite automata. This device was introduced by Holzer in [6] as multi-head finite automata with an additional constraint of obliviousness: the

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 273–283
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

trajectory of the heads only depends on the length of the input word. As emphasized in [6], such finite automata lead to significant computational power: they characterize parallel complexity NC^1 – which explains they can be efficiently parallelized. Their properties have been further discussed in [7]. We propose below a simulation of these data-independent multi-head finite automata by CA which gives rise to an efficient parallel speed-up.

This paper is organized as follows. The next section introduces the two models considered. Section 2 displays some of their features and abilities. Section 3 presents the simulation algorithm and its time and space cost.

1 Definitions

1.1 Multi-head finite automata

Given an integer $k \geq 1$, a two-way k-head finite automaton is a finite automaton reading an input word written between two end-markers using k heads that can move in any direction provided they do not go beyond these markers.

▶ **Definition 1.** A (deterministic) *two-way multi-head finite automaton* (2DFA(k) for short) is a tuple $(\Sigma, Q, \rhd, \lhd, q_0, q_\mathrm{a}, q_\mathrm{r}, k, \delta)$, where Σ is a finite set of *input symbols* (or *letters*), Q is a finite set of *states*, $\rhd \neq \lhd \notin \Sigma$ are the left and right *end-markers*, $q_0 \in Q$ is the *initial state*, $q_\mathrm{a} \neq q_\mathrm{r} \in Q$ are respectively the *accepting state* and the *rejecting state*, $k \geq 1$ is the *number of heads* and $\delta : Q \times (\Sigma \cup \{\rhd, \lhd\})^k \to Q \times \{-1, 0, 1\}^k$ the *transition function*; -1 means to move the head one letter to the left, 1 to move it one letter to the right and 0 to keep it on its current letter. For the heads to be unable to move beyond the end-markers, we require that if $\delta(q, a_1, \dots, a_k) = (q', m_1, \dots, m_k)$, then for any $i \in [\![1, k]\!]$, $a_i = \rhd \Rightarrow m_i \geq 0$ and symmetrically $a_i = \lhd \Rightarrow m_i \leq 0$.

A *configuration* of a 2DFA(k) on an input word w at a certain time $t \geq 0$ is a couple (p, q) where $p \in [\![0, |w| + 1]\!]^k$ is the position of the multi-head and q the current state.

The computation of a 2DFA(k) on an input word $w \in \Sigma^n$ starts with all heads on the left end-marker, and ends when the automaton reaches the accepting or the rejecting state. In the former case, the word is said to be accepted, while in the latter it is rejected. For some words, none of these cases happen and hence the automaton will enter a loop eventually. The language $L(\mathcal{F})$ *recognized* by a 2DFA(k) $\mathcal{F}$ is the set of the words accepted by $\mathcal{F}$. One can notice a 2DFA(k) necessarily enters a loop if it has not accepted nor rejected the input at step $|Q|(n + 2)^k$ steps, which is the number of configurations featuring w; so, we say the computation is over if we reach this step (it may of course take less).

We will focus now on the data-independent 2DFA (2DIDFA), a particular class of 2DFA for which the path followed by the heads only depends on the length of the input word, not on the letters thereof.

▶ **Definition 2.** Given $k \geq 1$, a 2DFA(k) $\mathcal{F}$ is said to be *oblivious* (or *data-independent*) if there exists a function $f_\mathcal{F} : \mathbb{N}^2 \to \mathbb{N}^k$ such that the position of its multi-head at time $t \in \mathbb{N}$ on any input word w is $f_\mathcal{F}(|w|, t)$.

1.2 Cellular automata

A cellular automaton is a parallel synchronous computing model consisting of an infinite number of finite automata called *cells* which are distributed on $\mathbb{Z}$ and share the same transition function, depending on the cell itself and its two neighbors.

▶ **Definition 3.** A *cellular automaton* is a tuple $(\Sigma, Q, \#, q_\mathrm{a}, q_\mathrm{r}, \delta)$, where Σ is the finite set of *input symbols* (or *letters*), $Q \supset \Sigma$ is the finite set of *states* and $\delta : Q^3 \to Q$ the *transition function.* $\# \in Q \setminus \Sigma$ is a particular quiescent state, verifying $\delta(\#, \#, \#) = \#$. $q_\mathrm{a} \neq q_\mathrm{r} \in Q$ are respectively the *accepting state* and the *rejecting state.* These are *persistent*, which means that a cell in such a state will never switch to another state: for any $q, q' \in Q$, $\delta(q, q_\mathrm{a}, q') = q_\mathrm{a}$ and $\delta(q, q_\mathrm{r}, q') = q_\mathrm{r}$.

A *configuration* is a function $\mathfrak{C} : \mathbb{Z} \to Q$. A *site* is a cell at a certain time step of the computation; $\langle c, t\rangle$ will denote the state of the site $(c, t) \in \mathbb{Z} \times \mathbb{N}$. The computation of a CA $\mathcal{C}$ on an input word w of size $n \geq 1$ starts at time 0 with all cells in state $\#$ except cells 1 to n where the letters of the word are written. This is the initial configuration $\mathfrak{C}_w$ associated to w. Then the cells update in parallel their respective states according to δ: for all $(c, t) \in \mathbb{Z} \times \mathbb{N}$, $\langle c, t + 1\rangle = \delta(\langle c - 1, t\rangle, \langle c, t\rangle, \langle c + 1, t\rangle)$.

The input word is accepted (resp. rejected) in time $t \in \mathbb{N}$ if and only if cell 1 enters the accepting state q_a (resp. the rejecting state q_r) at time t (and hence at any time $t' \geq t$). The language $L(\mathcal{C})$ *recognized* by the automaton is the set of the words it eventually accepts. $L(\mathcal{C})$ is said to be recognized in time $\tau : \mathbb{N} \to \mathbb{N}$ if and only if any word w is accepted or rejected in time $\tau(|w|)$.

2 Preliminaries

Our concrete question is the following: How long does it take to simulate a data-independent k-head finite automaton on CA? Regarding general multi-head finite automata, one can recall that a 2DFA(k) can be simulated on a deterministic multi-tape Turing machine in time $O(n^k)$, where n is the length of the input word [10]. Besides, it is well-known that CA are able to simulate in real time any deterministic (even multi-tape) Turing machine [9, 4]. A simple simulation consists in representing each tape of the Turing machine by two stacks – several stacks can be simulated simultaneously by a CA without loss of time. An upper bound in $O(n^k)$ for the time required by a CA to simulate a 2DIDFA(k) follows immediately. Now, how to reduce this time bound? As yet, no parallel speed-up is known to simulate Turing machines on CA; and no faster simulation of DIDFA on Turing machines taking the obliviousness constraint into account has been proposed. Here we will present a direct simulation which will take advantage of the oblivious feature of the DFA and so allow us to parallelize its computation.

2.1 Facts about multi-head finite automata

Let $\mathcal{F} = (\Sigma, Q, \rhd, \lhd, q_0, q_\mathrm{a}, q_\mathrm{r}, k, \delta)$ be a 2DIDFA, $n \geq 1$ be an integer and $w \in \Sigma^n$ be a word of size n. Let us look at the computation of $\mathcal{F}$ on input word w. The multi-head (composed of k heads) can be regarded as a device moving one point at a time in any direction within the set $\mathcal{W} = [\![0, n + 1]\!]^k$.

As $\mathcal{F}$ is data-independent, we can separate the path taken by the multi-head from the consecutive states of the automaton (depending on the letters of w). In other words, we can take a look at the path of the multi-head on an input word a^n, for any $a \in \Sigma$; it will be the same for w. If all heads are around the middle of the input, they will read only a for a long time and hence their movement will become periodic until one of them reaches an end-marker. That implies that the path of the multi-head is very simple as long as it is not near the *outer edge* $\mathcal{O} = \{p \in \mathcal{W} \mid \|p - \frac{n+1}{2}(1, \ldots, 1)\|_\infty = \frac{n+1}{2}\}$. To be more accurate, we can be sure the path has already become periodic precisely when the multi-head enters the central part of

$\mathcal{W}$ by crossing the *inner edge* $\mathcal{I} = \{p \in \mathcal{W} \mid \|p - \frac{n+1}{2}(1,\ldots,1)\|_\infty = \frac{n+1}{2} - |Q| - 1\}$ and it remains periodic until reaching the outer edge $\mathcal{O}$ (cf. Fig. 1).

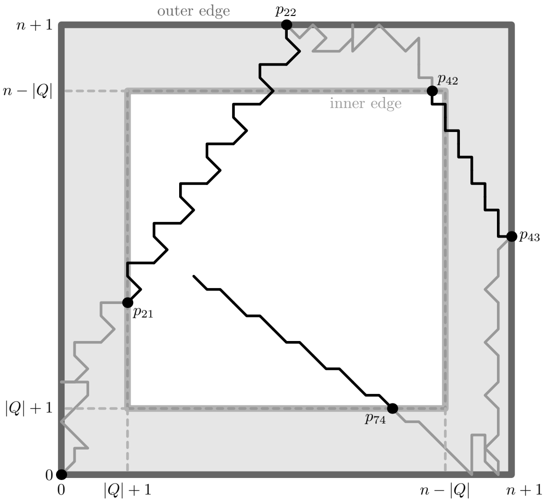


Figure 1 A representation of $\mathcal{W}$ for $k = 2$. The (beginning of the) path of the multi-head is drawn with the periodic sections (jumps) crossing the central white square (delimited by the inner edge $\mathcal{I}$) in black. For each jump a period shape is indicated in bold. The first five key points p_i that begin or end a jump are displayed as black dots.

The simulation of a 2DIDFA $\mathcal{F}$ on an input word w will involve the storing of specific configurations occurring in the run of $\mathcal{F}$ over the input word $a^{|w|}$ (see Fig. 1). We hence define the sequence of *key points* $(p_i, e_i, t_i)_i \in (\mathcal{W} \times Q \times \mathbb{N})^{\mathbb{N}}$ where (p_i, e_i) is the configuration at instant t_i of $\mathcal{F}$ over $a^{|w|}$, by $p_0 = (0,\ldots,0) \in \mathcal{O}$, $e_0 = q_0$, $t_0 = 0$ and for all $i > 0$,

- if $p_i \in \mathcal{I}$, p_{i+1} is the next position of $\mathcal{O}$ the multi-head encounters, at a certain time t_{i+1} with $\mathcal{F}$ in state e_{i+1}; this point is called a *jump* (across a periodic section);
- otherwise $t_{i+1} = t_i + 1$ and (p_{i+1}, e_{i+1}) is the next configuration of $\mathcal{F}$; this point is called a *step*.

Since the automaton is deterministic, any non-looping (accepting or rejecting) path cannot go through twice the same position in the same state. Thus, we can bound the number of jumps by $|Q||\mathcal{I}| \sim 2|Q|kn^{k-1}$. In the same way, as steps are located between the outer and inner edges, their number is bounded by $|Q|((n+2)^k - (n-2|Q|)^k) \sim 2|Q|(|Q|+1)kn^{k-1}$. The number of key points of a non-looping path is thus in $O(n^{k-1})$. In particular, it is linear when $k = 2$.

2.2 Basic techniques on cellular automata

A given computation of a CA can be easily represented by drawing successive configurations each one above its predecessor, forming a *space-time diagram*.

We will often have to perform several rather independent computations at the same time; this can easily be done by a “product” automaton which works with a finite number of *layers* supporting each a specific computation. Although rather independent, the layers can communicate between one another to exchange information, since any cell can see all layers. Typically, cells may be waiting for a firing squad to end on a layer before changing their behavior on another layer.

In the remainder of this article, we will have to handle computations involving coordinates. It is classical on CA to write integers in binary on segments of cells (one bit per cell) and execute basic operations with them. Here, all integers will be spelled backward on cells, the lowest bit being the leftmost one. For binary operators, we consider two integers superimposed on the same segment of cells.

It is easy to see that arithmetical operation such as *addition*, *subtraction* or *comparison* can be done in space and time linear in the number of bits of the operands. The same goes for *division by a fixed constant* where we consider the result to be both the quotient and the remainder.

Using the power of parallelism, it is also possible to achieve *multiplication* in linear space and time (see [1]). Moreover, we shall also use implicitly the fact that it is possible to synchronize any interval of cells in space and time linear in the size of this interval. This problem is often referred to as the *firing squad synchronization problem* (see [3, 8]).

In our construction, we shall use another basic operation that we call *selection*. The principle is the following: we fix an interval of cells of size n. Given an integer i (written in binary) between 0 and l positioned at the left of our interval, we want to select the i-th element of the interval and bring it at the right end of the interval (see Fig. 2).

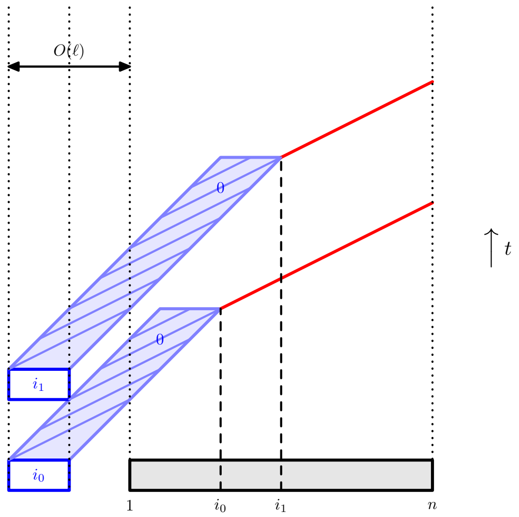


Figure 2 Schematic space-time diagram of a selection. The time scale of the picture has been compressed by factor 2, so that a signal of speed 1 (for instance, in red) seems to be of speed 2.

The basic principle of this operation is quite simple, we shift the integer i along the interval and at each shift decrease its value by one. When reaching 0, we take the content of the corresponding cell inside the interval and send it at maximal speed toward the end of the interval [5]. Thus, the shift speed can be constant and the whole selection achieved in linear time using as workspace only the constant logarithmic space that is required to write the value l.

In addition, this operation can be pipelined. That is, if we suppose that for a fixed interval, we have m integers written as ℓ-bit strings at the beginning of the interval every $\ell' = O(\ell)$ time steps, then the total time for achieving all the selections is $m\ell' + n$.

3 Simulation

▶ **Theorem 4.** *Given $k > 1$, for any $2DIDFA(k)$ $\mathcal{F}$ recognizing a language $\mathcal{L}$, there exists a CA $\mathcal{C}$ recognizing $\mathcal{L}$ in time $O(n^{k-1}\log(n))$ and space $O(n^{k-1})$, where n is the size of the input word.*

The rest of this paper will be devoted to the proof of this theorem. We will only consider the case $k = 2$ to alleviate the descriptions, but it should be straightforward to generalize this proof.

We assume now that we have a 2DIDFA(2) $\mathcal{F} = (\Sigma,\ Q, \triangleright, \triangleleft, q_0, q_\mathrm{a}, q_\mathrm{r}, 2, \delta)$. Let us take an arbitrary input word $w \in \Sigma^n$, given an integer $n \geq 2|Q|$. We will define a CA $\mathcal{C} = (\Sigma, Q', \#, q'_\mathrm{a}, q'_\mathrm{r}, \delta')$ fulfilling the requirements of the theorem. Instead of giving the full description of its state set and transition function, we will describe its behavior when given w as input (the finite number of words that are too short are treated by the CA as a particular case, we can hence forget them).

The execution of $\mathcal{C}$ on w involves coordinates. They all range from 0 to $n + 1$, hence we need $\ell(n) = \lceil \log_2(n + 2)\rceil$ bits to be able to write any of them. We have to compute it, thus the first thing $\mathcal{C}$ will do is to write $n + 1$ in binary on cells $[\![1, \ell(n)]\!]$. This can be done in time $O(n)$.

3.1 Computation of the sequence of key points

We have, as defined previously, $\mathcal{W} = [\![0, n + 1]\!]^2$, $\mathcal{O} = (\{0, n + 1\} \times [\![0, n + 1]\!]) \cup ([\![0, n + 1]\!] \times \{0, n+1\})$, $\mathcal{I} = (\{|Q|+1, n-|Q|\} \times [\![|Q|+1, n-|Q|]\!]) \cup ([\![|Q|+1, n-|Q|]\!] \times \{|Q|+1, n-|Q|\})$ and the sequence of key points $(p_i, e_i, t_i)_i$ summing up the computation over a^n. What we want to do now is to output these key points in order on the CA. Note that the coordinates x_i, y_i of position p_i, time t_i, and index i are polynomial in n; thus, they can be encoded in logarithmic space, while state e_i lies in a finite set, only requiring a single cell. Each such point will be written as superimposed $\ell(n)$-bit strings (for x_i, y_i, t_i, and i) together with e_i.

We will compute the sequence of key points iteratively:

1. The procedure is initiated from the first key point $((0, 0), q_0, 0)$.
2. At the start of iteration i, key point $(p_i = (x_i, y_i), e_i, t_i)$ is given. First we determine if $p_i \in \mathcal{I}$. It consists in checking whether at least one coordinate of p_i is equal to $|Q| + 1$ or $n - |Q|$.
   - *if* $p_i \notin \mathcal{I}$ (case of a step). We check for each head whether it lies on the outer edge (*i.e.*, whether $x_i, y_i \in \{0, n + 1\}$). This indicates which letter is read ($\triangleright$,$\triangleleft$ or a). According to this information, we mimic a single transition of the automaton. Namely, we compute the coordinates of p_{i+1} and t_{i+1} by means of some increment or decrement, along with the next state e_{i+1}.
   - *if* $p_i \in \mathcal{I}$ (case of a jump). Between p_i and p_{i+1}, the automaton follows a periodic behavior that only depends on the current state e_i. Such behavior can be specified by finite parameters, namely its period $r_i = (u_i, v_i) \in [\![-|Q|, |Q|]\!]^2$ and its shape s_i (*i.e.*, the head's sequence of moves in $\{-1, 0, 1\}^2$, of length at most $|Q|$). Since the number of states and so the number of distinct periodic behaviors are finite, we can assume that r_i and s_i are available in due time. Thus, we perform operations $(n - x_i)/u_i$ and $(n - y_i)/v_i$ to get the respective quotients a_i and b_i and remainders h_i and k_i. We select the smaller quotient $c_i \in \{a_i, b_i\}$ and then compute $a'_i = u_i \times c_i$ and $b'_i = v_i \times c_i$. The next key point is $p_{i+1} = (x_i + a'_i, y_i + b'_i)$. To be accurate, we have to add the remainder of the period depending on h_i (if $c_i = a_i$) or k_i (if $c_i = b_i$) and on the particular

shape s_i of the period. Although this remainder can have negative coordinates in some particular cases, their absolute values are always bounded by $|Q|$.

3. The procedure stops if either $e_{i+1} \in \{q_{\mathrm{a}}, q_{\mathrm{r}}\}$ or a loop is detected. In this last case, we check the number of key points already computed and the elapsed time. If $i+1 = |Q|((n+2)^2 - (n-2|Q|-2)^2)$ or $t_{i+1} > |Q|(n+2)^2$, we know that the automaton has entered a loop and we can thus definitely stop the simulation.

What is the cost of the whole iteration procedure? Each iteration performs only a finite number of linear operations over integers of size $\ell(n) = O(\log(n))$ and thus is done in space and time $O(\log(n))$. Since the number of key points for a non-looping computation is in $O(n)$, the whole procedure takes $O(n \log(n))$ time steps. One can notice it is conducted very slowly, with $O(\log(n))$ time steps to get only one move of the multi-head in case of step points. But we save a lot of time with every jump across a periodic section, computing $O(n)$ moves of the multi-head within $O(\log(n))$ time steps of the CA.

3.2 Computation of the states

For the moment, we have computed the sequence of key positions $(p_i)_i$ that the multi-head would follow on input word w, but we still do not know its successive states (we have seen only those corresponding to input word a^n). A fortiori we do not know whether w should be accepted. We are now about to get past this lack. What we want to do is to compute, for all key positions p_i, the function $\delta_i \in Q^Q$ such that for all $q \in Q$, if at some time step t the DFA is in state q (for input word w) with the multi-head on key position p_i, then at time $t + t_{i+1} - t_i$ it is in state $\delta_i(q)$ with the multi-head on key position p_{i+1}. One can notice that this way $\delta_i(q)$ may be undefined if state q does not lead to the actual path between p_i and p_{i+1}. If so, it is no problem, we just set $\delta_i(q) = \bullet \notin Q$.

To compute these functions, we have two cases according to whether key position p_i is in $\mathcal{I}$ or not. If it is a step ($p_i \notin \mathcal{I}$), its associated function δ_i only performs a single transition of the DFA and so can be simply computed from the letter lying at this position. In case of a jump ($p_i \in \mathcal{I}$), the problem is more complex, since the associated function mimics all successive transitions performed from p_i to p_{i+1}. But, making use of the regularities of oblivious computation, we can compute simultaneously all these jump functions in linear time.

Pre-computation of all feasible jump functions

Each cell $c \in \{|Q|+1, n-|Q|\}$ will compute the jump function associated to potential key position $p = (c, |Q|+1)$ (and at the same time $(c, n-|Q|)$ and $(|Q|+1, c)$ and $(n-|Q|, c)$). First we will assume that the DFA is in some state q and that the forthcoming periodic trajectory is of shape s.

Cell c plays the role of the multi-head. Once every two time steps, it will update the state of the DFA according to the letters it reads and send two signals at speed 1, one toward the left and one toward the right, to tell the rest of the cells supporting the input word what is the next move of the multi-head. Two copies of the input word are shifted according to the signals received and hence this cell has access to the letters encountered by the multi-head. Because of the duration of the transmission of the order to shift, we need to have parts of the copies provisionally compressed (two letters may lie on the same cell) or dilated, some cells being empty (cf. Fig 3). This process (for particular values q and s) is conducted until an end-marker is fed to cell c, or until it realizes state q does not induce a period of shape s. In both cases, if $p = p_i$ for some i, we have got $\delta_i(q)$ after $O(n)$ time steps.

What we just described grants us state $\delta_i(q)$ only for the key positions at the four corners of the inner edge. Anyway, there is a nice way to get all the other functions simultaneously. Suppose p_i is at a corner of $\mathcal{I}$; how do we compute, for instance, $\delta_j(q)$ for $p_j = (x_i + h, y_i) \in \mathcal{I}$? The two points share their first coordinate (every $p \in \mathcal{I}$ shares a coordinate with a corner), thus for a same periodic behavior the two trajectories of the multi-head are identical, except for the first head, which is shifted by h letters. $\delta_i(q)$ is computed on cell x_i from some time step $t \in \mathbb{N}$; this cell will send each second-head letter it reads at speed 1 to the other cells. Thus, from time step $t + h$, cell $x_i + h$ will receive the correct letters of the second head, while the copy of the tape for the first head is shifted on this cell with the same delay of h time steps (cf. Fig. 3). $\delta_j(q)$ can in this way be computed on cell $x_i + h$.

This previous process is actually achieved simultaneously for every $q \in Q$. Finally, using one layer for each possible shape, we can have all feasible jump functions written on segment $[\![|Q| + 1, n - |Q|]\!]$ in $O(n)$ time steps.

Selecting the transition function associated to a key point

Suppose we are given all the feasible jump functions written on segment $[\![|Q| + 1, n - |Q|]\!]$, a copy of the input written on cells $[\![1, n]\!]$ and a key point initially written in position $[\![-2\ell(n) + 1, -\ell(n)]\!]$. Using those data, we want to retrieve the function associated to the key point. Two cases are considered depending on whether the function performs a single transition or a jump transition of the DFA.

- Case of step points: according to the coordinates of the step position, we collect the input letters read by each head of the DFA and then deduce the proper function. This is done in time and space $O(n)$.
- Case of jump points: making use of the operation of selection described in section 2.2, we select the jump function written on cell c and layer l, where c is specified by the position and l (depending on the period) by the state of the jump point. This is done in time and space $O(n)$.

3.3 Final stage

Compiling the previous procedures, the cellular automaton $\mathcal{C}$ works the following way:

- First, $\mathcal{C}$ pre-computes all the feasible jump functions and writes them on cells $[\![|Q| + 1, n - |Q|]\!]$. This is done in both space and time $O(n)$.
- Then, $\mathcal{C}$ generates the list of key points. This is done in time $O(n \log(n))$ and space $O(\log(n))$.
- As soon as one key point is written, $\mathcal{C}$ selects its associated function. Each selection is done in time and space $O(n)$.
- Once a new selection is over, $\mathcal{C}$ updates the current state of the automaton $\mathcal{F}$ according to the proper function. In case it is the accepting or the rejecting state, or if the time of the last key point reaches the maximal number of moves the multi-head is supposed to perform, $\mathcal{C}$ terminates its computation.

These methods obviously work in space $O(n)$. In regard to the time, since the selections are pipelined, the bound corresponds to the time required to generate all the key points plus the time of the last selection. Hence the total running time is in $O(n \log(n)) + O(n)$, *i.e.* in $O(n \log(n))$, leading to the theorem. ◀

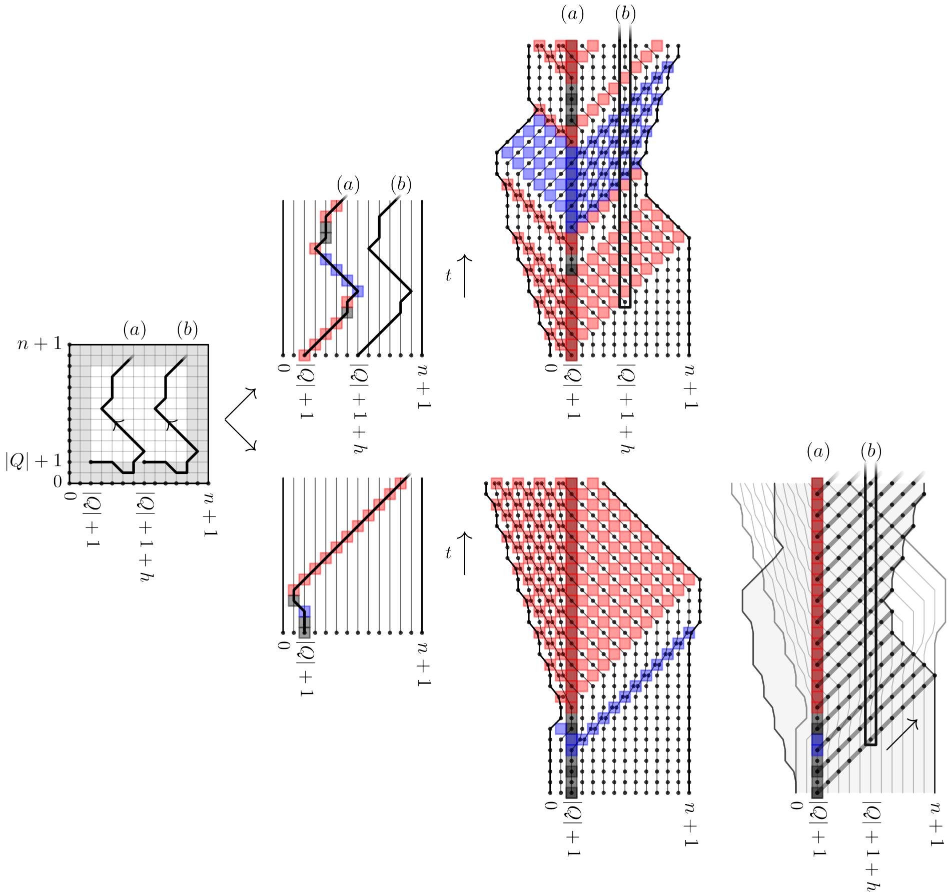


Figure 3 Simulation of the multi-head from a corner of the inner edge. On the left we have two portions of the path (a) and (b) that are translated copies of each other by h letters. Portion (a) starts at corner $(|Q|+1, |Q|+1)$ and is simulated by cell $|Q|+1$ (the darkened one) by the CA (on the right) with two layers, one by head. On each layer, according to the state of the DFA, the dark cell may send signals telling the other cells to shift their letter to the left (blue signal) or to the right (red signal). If they see no signal, the cells keep their letter, which are symbolized by dots, linked to indicate where they are moved at each time step – end-markers are also represented by dots, linked in black. As the shifted tapes on each layer cross the dark cell, the latter gets the appropriate letters within two time steps to deduce the next move and state of the DFA. Cell $|Q|+1+h$ simulates portion (b) h time steps later. It sees directly the correct letters on the first layer while those for the second head are sent by the dark cell.

Conclusion

We have presented an efficient construction that simulates oblivious k-head finite automata on cellular automata in time $O(n^{k-1}\log(n))$ and space $O(n^{k-1})$. Such simulations achieving parallel speed-up are scarce. The performance gain is in $O(n/\log(n))$ as regards the naïve simulation without speed-up, which processes the same task in time $O(n^k)$.

Our result fully exploits the oblivious feature of the sequential computation. Now, it is another challenge to achieve parallel speed-up for multi-head finite automata without the constraint of data-independence.

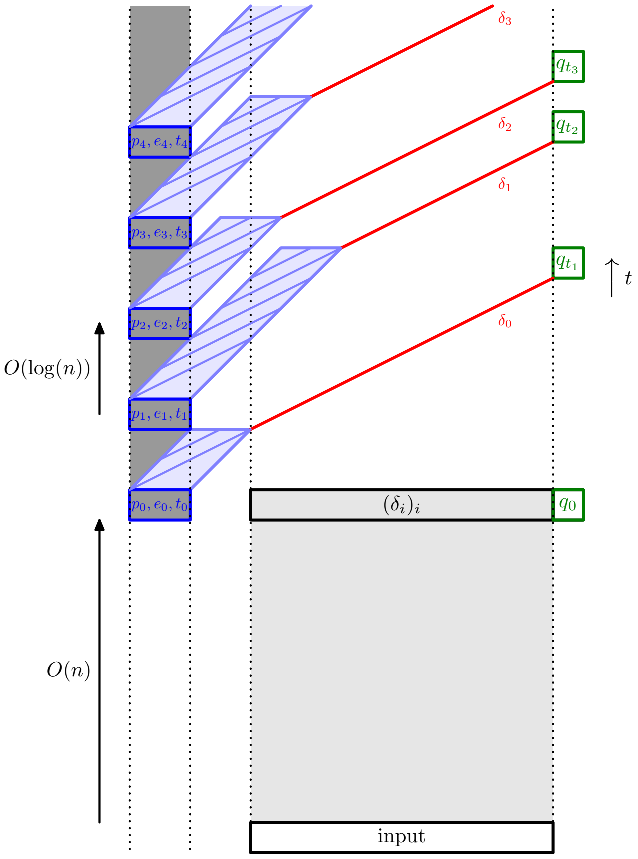


Figure 4 Global simulation. Light grey corresponds to the pre-computation of jumps functions, dark grey to the computation of the sequence of key points, and the rest is the selection (here again, the time scale is compressed by factor 2).

References

1 A. J. Atrubin. A one-dimensional real-time iterative multiplier. *IEEE Transactions on Electronic Computers*, 14(1):394–399, 1965.

2 S. N. Cole. Real-time computation by n-dimensional iterative arrays of finite-state machines. *IEEE Transactions on Computers*, 18(4):349–365, 1969.

3 K. Čulík II. Variations of the firing squad problem and applications. *Information Processing Letters*, 30(3):152–157, 1989.

4 M. Delorme. An introduction to cellular automata. In M. Delorme and J. Mazoyer, editors, *Cellular Automata: A Parallel Model*, pages 5–50. Kluwer Academic Publishers, 1997.

5 M. Delorme and J. Mazoyer. Algorithmic tools on cellular automata. In G. Rozenberg, T. Baeck, and J. Kok, editors, *Handbook of Natural Computing*. Springer, Berlin, to appear.

6 M. Holzer. Multi-head finite automata: Data-independent versus data-dependent computations. *Theoretical Computer Science*, 286(1):97–116, 2002.

7 M. Holzer, M. Kutrib, and A. Malcher. Multi-head finite automata: Characterizations, concepts and open problems. In T. Neary, D. Woods, A. K. Seda, and N. Murphy, editors, *The Complexity of Simple Programs (CSP'08)*, EPTCS, pages 93–107, 2008.

8 J. Mazoyer. On optimal solutions to the firing squad synchronization problem. *Theoretical Computer Science*, 168(2):367–404, 1996.

9 A. R. Smith III. Simple computation-universal cellular spaces. *Journal of the ACM*, 18(3):339–353, 1971.

10 K. Wagner and G. Wechsung. *Computational Complexity*. D. Reidel, Dordrecht, 1986.

Stochastic Cellular Automata Solve the Density Classification Problem with an Arbitrary Precision

Nazim Fatès

INRIA Nancy – Grand Est, LORIA, Nancy Université
Campus scientifique, BP 239, 54 506 Vandœuvre-lès-Nancy, France
nazim.fates@loria.fr

Abstract

The density classification problem consists in using a binary cellular automaton (CA) to decide whether an initial configuration contains more 0s or 1s. This problem is known for having no exact solution in the case of binary, deterministic, one-dimensional CA. Stochastic cellular automata have been studied as an alternative for solving the problem. This paper is aimed at presenting techniques to analyse the behaviour of stochastic CA rules, seen as a "blend" of deterministic CA rules. Using analytical calculations and numerical simulations, we analyse two previously studied rules and present a new rule. We estimate their quality of classification and their average time of classification. We show that the new rule solves the problem with an arbitrary precision. From a practical point of view, this rule is effective and exhibits a high quality of classification, even when the simulation time is kept small.

1998 ACM Subject Classification F.1.1 Models of Computation, G.3 Probability and Statistics

Keywords and phrases stochastic and probabilistic cellular automata, density classification problem, models of spatially distributed computing, stochastic process

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.284

Introduction

The density classification problem is one of the most studied inverse problems in the field of cellular automata. Informally, it requires that a binary cellular automaton — or more generally a discrete dynamical system — decides whether an initial binary string contains more 0s or more 1s. In its classical formulation, the cells are arranged in a ring and each cell can only read its own state and the states of the neighbouring cells. The challenge is to design a behaviour of the cells that drives the system to a uniform state, that consists of all 1s if the initial configuration contained more 1s and all 0s otherwise. In short, the convergence of the cellular automaton should decide whether the initial density of 1s was greater or lower than 1/2.

Although the task looks trivial, it has attracted a considerable amount of research since its formulation by Packard [13]. The difficulty of finding a solution comes from the impossibility to centralise the information or to use any classical counting technique. Instead, the convergence to a uniform state should be obtained by using only *local* decisions, that is, by using an information that is limited to the close neighbours of a cell. Moreover, as CA are homogeneous by nature (the cells obey the same law), there can be no specialisation of the cells for a partial computation. Solving the problem efficiently requires to find the right balance between deciding locally with a short-range view and following other cells' decision to attain a global consensus.

The quest for efficient rules has been conducted on two main directions: man-designed rules and rules obtained with large space exploration techniques such as genetic algorithms

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 284–295
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

(*e.g.*, [11]). The Gacs-Kurdymov-Levin (GKL) rule, which was originally designed in the purpose of resisting small amounts of noise [14, 5], proved to be a good candidate (∼80% of the initial conditions well-classified on rings of 149 cells) and remained unsurpassed for a long time. In 1995, after observing that outperforming this rule was difficult, Land and Belew issued a key result: no perfect (deterministic) density classifier that uses only two states exist [9]. However, this did not stop the search for efficient CA as nothing was known about how well a rule could perform. In particular, it was asked whether an upper bound on a rule quality would exist. The search for rules with an increasing quality has been carried on until now, with genetic algorithms as the main investigation tool (see *e.g.* [4, 12] and references therein).

On the other hand, various modifications to the classical problem were proposed, allowing one to solve the problem exactly. For instance, Capcarrere *et al.* proposed to modify the output specification of the problem to find a solution that classifies the density perfectly [3]. Fukś showed that running two CA rules successively would also provide an acceptable solution [7]. This issue was further explored by Martins and Oliveira, who discovered various couples and triples of rules that solve the problem when applied sequentially and for a given number of steps [10]. Some authors also proposed to embed a memory in the cells, which is another method for enhancing the abilities of the rules [1, 16].

However, all of these solutions break the original specification of the problem, where the cells have only two states and obey a homogeneous rule. The use of stochastic (or probabilistic)[1] CA is an interesting alternative that complies with these two conditions. Indeed, in stochastic CA, the only modification to the CA structure is that the outcome of the local transitions of the cells is no longer deterministic: it is specified by a probability to update to a given state. This research path was opened by Fukś who exhibited a rule which acts as a "stochastic copy" of the state of the neighbouring cells [8]. However, this mechanism generates no force that drives the system towards its goal; the convergence is mainly attained with a random drift of the density (see Sec. 2). Recently, Schüle et al. proposed a stochastic rule that implements a local majority calculus [15]. This allows the system to converge to its goal more efficiently, but the convergence rates still remain bounded by some intrinsic limitations (see Sec. 3).

We propose to follow this path and present a new stochastic rule that solves the density classification problem with an arbitrary precision, that is, with a probability of success arbitrarily close to 1. This result answers negatively the open question to whether there exists an upper bound on the success rate one can reach. The idea is to use randomness to solve the dilemma between the local majority decisions and the propagation of a consensus state. A trade-off is obtained by tuning a single parameter, η, that weights two well-known deterministic rules, namely the majority rule and the "traffic" rule. We show that the probability of making a good classification approaches 1 as η is set closer to 0. To evaluate the "practical" use of our rule, we perform numerical simulations. Results show that this rule attains qualities of classification that have been out of reach so far.

1 Formalisation of the Problem

In this section, we define the deterministic Elementary Cellular Automata and their stochastic counterpart. We introduce the main notations for studying our problem.

[1] Both terms 'stochastic' and 'probabilistic' CA are found in literature. We prefer to employ the former as etymologically the Greek word 'stochos' implies the idea of goal, aim, target or expectation.

1.1 Elementary Cellular Automata

Let $\mathcal{L} = \mathbb{Z}/n\mathbb{Z}$ represent a set of n cells arranged in a ring. Each cell can hold a state in $\{\mathtt{0}, \mathtt{1}\}$ and we call a *configuration* the state of the system at a given time ; the configuration space is $\mathcal{E}_n = \{\mathtt{0}, \mathtt{1}\}^{\mathcal{L}}$, it is finite and we have $|\mathcal{E}_n| = 2^n$. We denote by $|x|_{\mathrm{P}}$ the number of occurrences of a pattern P in x. The *density* $\rho(x)$ of a configuration $x \in \mathcal{E}_n$ is the ratio of 1s in this configuration: $\rho(x) = |x|_{\mathtt{1}}/n$. We denote by $\mathbf{0} = \mathtt{0}^{\mathcal{L}}$ and $\mathbf{1} = \mathtt{1}^{\mathcal{L}}$ the two special uniform configurations. For $q \in \{\mathtt{0}, \mathtt{1}\}$, a configuration x is a *q-archipelago* if all the cells in state q are isolated, *i.e.*, if x does not contain two adjacent cells in state q.

In all the following, we assume that n is odd. This will prevent us from dealing with configurations that have an equal number of 0s and 1s.

An *Elementary Cellular Automaton* (ECA) is a one-dimensional binary CA with nearest neighbour topology, defined by its *local transition rule*, a function $\phi : \{\mathtt{0}, \mathtt{1}\}^3 \to \{\mathtt{0}, \mathtt{1}\}$ that specifies how to update a cell using only nearest-neighbour information. For a given ring size n, the *global transition rule* $\Phi : \mathcal{E}_n \to \mathcal{E}_n$ associated to ϕ is the function that maps a configuration x^t to a configuration x^{t+1} such that:

$$\forall c \in \mathcal{L},\, x_c^{t+1} = \phi(x_{c-1}^t, x_c^t, x_{c+1}^t)$$

A *Stochastic Elementary Cellular Automaton* (sECA) is also defined by a local transition rule, but the next state of a cell is known only with a given probability. In the binary case, we define $f : \{\mathtt{0}, \mathtt{1}\}^3 \to [0, 1]$ where $f(x, y, z)$ is probability that the cell updates to state 1 given that its neighbourhood has the state (x, y, z). The *global transition rule* F associated to the local function f is the function that assigns to a random configuration x^t the random configuration x^{t+1} characterised[2] by:

$$\forall c \in \mathcal{L},\, x_c^{t+1} = \mathcal{B}_c^t \left(f(x_{c-1}^t, x_c^t, x_{c+1}^t) \right) \tag{1}$$

where x_c^t denotes the random variable that is given by observing the state of cell c at time t and where $(\mathcal{B}_c^t)_{c \in \mathcal{L}, t \in \mathbb{N}}$ is a series of independent Bernoulli random variables, *i.e.*, $\mathcal{B}_c^t(p)$ is a random variable that equals to 1 with probability p and 0 with probability $1 - p$.

1.2 Density Classifiers

We say that a configuration x is a *fixed point* for the global function F if we have $F(x) = x$ with probability 1 and that F is a *(density) classifier* if $\mathbf{0}$ and $\mathbf{1}$ are its two only fixed points.

For a classifier $\mathcal{C}$, let $T(x)$ be the random variable that takes its values in $\mathbb{N} \cup \infty$ defined as:

$$T(x) = \min \left\{ t : x^t \in \{\mathbf{0}, \mathbf{1}\} \right\}$$

We say that $\mathcal{C}$ *correctly classifies* a configuration x if $T(x)$ is finite and if $x^{T(x)} = \mathbf{1}$ for $\rho(x) > 1/2$ and $x^{T(x)} = \mathbf{0}$ for $\rho(x) < 1/2$. The *probability of good classification* $\mathsf{G}(x)$ of a configuration x is the probability that $\mathcal{C}$ correctly classifies x.

To evaluate quantitatively the quality of a classifier requires to choose a distribution of the initial configurations. Various such distributions are found in literature, often without

[2] Note that defining rigorously the series of random variables x^t obtained from F would require to introduce advanced tools from the probability theory. In particular, one should define a space of realisation Ω and always consider probability measures on Ω and for all $\omega \in \Omega$, define the random variables with respect to the configurations $x^t(\omega) \in \mathcal{E}_n$. For the sake of simplicity, and as it is frequently done, the parameter ω is omitted and the random variables are defined only with regard to their probability of realisation on Ω.

an explicit mention, and this is why one may read different quality evaluations for the same rule (for instance compare the results given for the GKL rule: 82% in Ref. [3] and 97.8% in Ref. [9]). In order to avoid ambiguities, we re-define here the three main distributions of initial configurations that have been used by authors:

(a) The *binomial* distribution μ_b is obtained by choosing a configuration uniformly in $\mathcal{E}_n$.

(b) The *d-uniform* distribution μ_d is obtained by choosing an initial probability p uniformly in $[0,1]$ and then building a configuration by assigning to each cell a probability p to be in state $\mathtt{1}$ and a probability $1-p$ to be state $\mathtt{0}$.

(c) The $\mathtt{1}$-*uniform* distribution $\mu_\mathtt{1}$ is obtained by choosing a number k uniformly in $\{0,\dots,n\}$ and then by choosing uniformly a configuration in the set of configurations of $\mathcal{E}_n$ that contain exactly k ones.

Formally,

$$\forall x \in \mathcal{E}_n, \quad \mu_\mathrm{b}(x) = \frac{1}{2^n} \quad ; \quad \mu_\mathrm{d}(x) = \int_0^1 p^k(1-p)^{n-k}dp \quad ; \quad \mu_\mathtt{1}(x) = \frac{1}{n+1}\cdot\frac{1}{\binom{n}{k}}$$

where $k = |x|_\mathtt{1}$ is the number of $\mathtt{1}$s in x.

▶ Proposition 1. The d-uniform distribution μ_d and the $\mathtt{1}$-uniform distribution $\mu_\mathtt{1}$ are equivalent.

This equality can be established by identifying $\mu_\mathrm{d}(x)$ to the values of the so-called 'Beta function' or 'Euler's integral of first kind'.

Given a ring size n, a distribution μ on $\mathcal{E}_n$, the *quality* Q of a classifier $\mathcal{C}$ is defined by:

$$Q(n) = \sum_{x\in\mathcal{E}_n} \mathsf{G}(x)\cdot\mu(x)$$

In this paper, we evaluate the performance of a classifier using a binomial quality Q_b, defined with the distribution μ_b and a d-uniform quality Q_d, defined with the distribution μ_d. Intuitively, we see that for most classifiers, we will have $Q_\mathrm{d} > Q_\mathrm{b}$. Indeed, when we take the binomial distribution, as n grows to infinity, most initial configurations of $\mathcal{E}_n$ have a density close to 1/2 and are generally more difficult to classify that configuration with densities close to 0 or 1. The d-uniform distribution avoids this difficulty by assigning an equal chance to appear to all the initial densities.

Similarly, we define the average classification time with regard to distribution μ as

$$T_\mu = \sum_{x\in\mathcal{E}_n} \mathbb{E}\{\,T(x)\,\}\mu(x)$$

We denote by T_b and T_d the average classification time obtained with the μ_b and μ_d distributions, respectively. As, for most classifiers, we have $T_\mathrm{d} < T_\mathrm{b}$, we are only interested in estimating T_b.

1.3 Structure of the sECA space

Obviously, the classical deterministic ECA are particular sECA with a local rule that takes its values in $\{0,1\}$. The space of sECA can be described as an eight-dimensional hypercube with the 256 ECA in its corners. This can be perceived intuitively if we see sECA rules as points of a hypercube, to which we apply the operations of addition and multiplication. More formally, taking k sECA $f_1,\dots,f_k$ and $w_1,\dots,w_k$ real numbers in $[0,1]$ such that $\sum_{i=0}^{k} w_i = 1$, the barycenter of the sECA (f_i) with weights w_i is the sECA g defined with:

$$\forall x,y,z \in \{\mathtt{0},\mathtt{1}\},\; g(x,y,z) = \sum_{i=0}^{k} w_i.f(x,y,z)$$

Table 1 Table of the 8 active transitions and their associated letters. The transition code of an ECA is the sequence of letters of its active transitions.

A	B	C	D	E	F	G	H
000 1	001 1	100 1	101 1	010 0	011 0	110 0	111 0

As a consequence, one may choose to express an sECA as a barycenter of other ECA. The most intuitive basis of the sECA space is formed by the 8 ECA that have only one transition that leads to 1: the coordinates correspond to the values $f(x, y, z)$. Equally, one may express a sECA as a barycenter of the 8 (deterministic) ECA that have only one *active* transition, *i.e.*, only one change of state in their transition table. Such ECA are labelled A, B, ..., H according to the notation introduced in Ref. [6] and summed up in Tab. 1. Formally, for every sECA f, there exists a 8-tuple $(p_\mathrm{A}, p_\mathrm{B}, \dots, p_\mathrm{H}) \in [0, 1]^8$ such that: $f = p_\mathrm{A} \cdot \mathsf{A} + p_\mathrm{B} \cdot \mathsf{B} + \cdots + p_\mathrm{H} \cdot \mathsf{H}$. We denote this relationship by $f = [p_\mathrm{A}, p_\mathrm{B}, \dots, p_\mathrm{H}]_\mathrm{T}$, where the subscript T stands for (active) *transitions*.

This basis presents many advantages for studying the random evolution of configurations (see Ref. [6]). For instance, the group of symmetries of a rule can easily be obtained: the left-right symmetry permutes p_B and p_C, and p_F and p_G, whereas the 0-1 symmetry permutes p_A and p_H, p_B and p_G, etc.

This *transition code* also allows us to easily write the conservation laws of a stochastic CA and to estimate some aspects of its global behaviour. To do this analysis, we write $a(x) = |x|_{000}$, $b(x) = |x|_{001}$, ..., $h(x) = |x|_{111}$ (see Tab. 1) and drop the argument x when there is no ambiguity. The following equalities hold [6]:

$$b + d = e + f = c + d = e + g \quad ; \quad b = c \quad ; \quad f = g \tag{2}$$

We now detail how to use these tools to analyse the behaviour of an sECA.

2 Fukś Density Classifier

To start examining how stochastic CA solve the density classification problem, let us first consider the probabilistic density classifier proposed by Fukś [8]. For $p \in (0, 1/2]$, the local rule $\mathbf{C}_1$ is defined with the following transition table:

(x, y, z)	000	001	010	011	100	101	110	111
$f(x, y, z)$	0	p	$1-2p$	$1-p$	p	$2p$	$1-p$	1

For any ring size n, this rule is a density classifier as $\mathbf{0}$ and $\mathbf{1}$ are its only fixed points. With the transition code of Sec. 1.3, we write:

$$\begin{aligned} \mathbf{C}_1 &= [0, p, p, 2p, 2p, p, p, 0]_\mathrm{T} \\ &= p \cdot \mathsf{BDEG} + p \cdot \mathsf{CDEF} \end{aligned}$$

where the rules[3] BDEG($\mathbf{170}$) and CDEF($\mathbf{240}$) are the left and right shift respectively. This means that Fuks' rule can be interpreted as applying, for each cell independently: (a) a left

[3] We give the "classical" rule code into parenthesis ; it is obtained by converting the series of 8 bits of the transition table (000 to 111) to the corresponding decimal number.

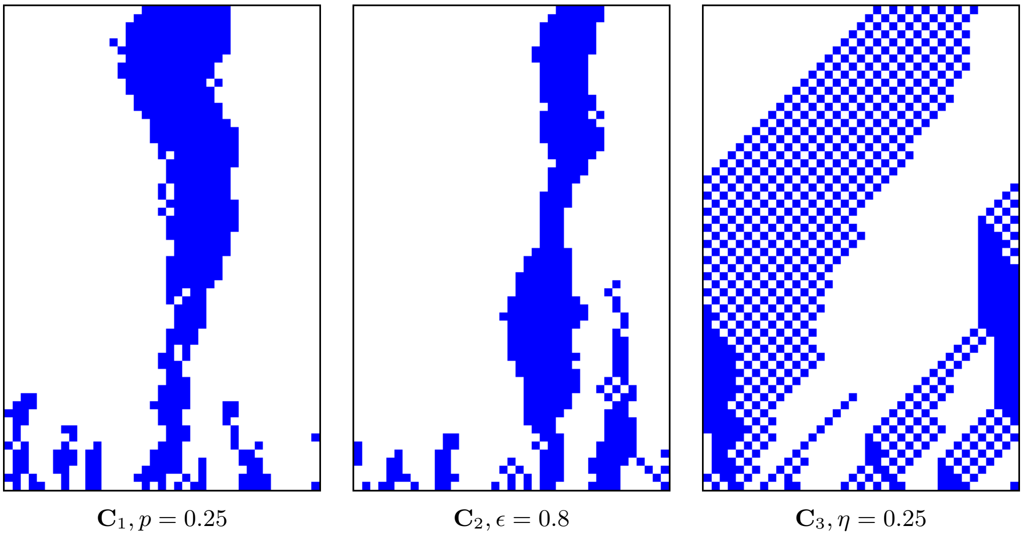


Figure 1 Space-time diagrams showing the evolution of the $\mathbf{C}_1, \mathbf{C}_2, \mathbf{C}_3$ classifiers with $n = 39$, and same initial density ~ 0.4. Time goes from bottom to top ; white cells are 0-cells and blue cells are 1-cells. (left & middle) evolution will most probably end with a good classification ($\mathbf{0}$); (right) evolution will with end with a good classification with probability 1 (an archipelago has been reached).

shift with probability p, (b) a right shift with probability p, and (c) staying in the same state with probability $1 - 2p$ (see Fig. 1). We also note that this rule is invariant under both the left-right and the 0-1 permutations (as $p_\mathrm{B} = p_\mathrm{C} = p_\mathrm{F} = p_\mathrm{G}$, $p_\mathrm{A} = p_\mathrm{H}$ and $p_\mathrm{D} = p_\mathrm{E}$).

▶ **Theorem 1.** *For the classifier $\boldsymbol{C}_1$ set with $p \in (0, 1/2]$,*

$$\forall x \in \mathcal{E}_n,\ \mathsf{G}(x) = \max\ \{\rho(x), 1 - \rho(x)\} \quad \textit{and} \quad T_\mathrm{b} \leq \frac{1}{4p} \cdot n^2$$

The relationship on $\mathsf{G}(x)$ was observed experimentally with simulations and partially explained by combinatorial arguments [8]. As for the classification time of the system, no predictive law was given. We now propose a proof that uses the analytical tools developed for asynchronous ECAs [6] and completes the results established by Fukś. The proof stands on the following lemma:

▶ **Lemma 2.** *For a sequence of random variables $(x^t)_{t \in \mathbb{N}}$ that describes the evolution of a stochastic CA with the initial condition $x \in \mathcal{E}_n$, let M be a mapping $M : \mathcal{E}_n \to \{0, \ldots, m\}$ where m is any integer, and let (X_t) be the sequence of random variables defined by $\forall t, X_t = M(x^t)$. If X_t and $\Delta X_{t+1} = X_{t+1} - X_t$ verify that:*

- *the stochastic process (X_t) is a martingale on $\{0, \ldots, m\}$, that is, for a filtration $\mathcal{F}_t$ adapted to (X_t), $\mathbb{E}\big\{\, \Delta X_{t+1}\, |\mathcal{F}_t \,\big\} = 0$,*
- *$X_t \in \{1, \ldots, m-1\} \implies \mathrm{var}\big\{\, \Delta X_{t+1} \,\big\} > v$,*

then:

$$\Pr\{X_T = m\} = \frac{q}{m}$$

and the absorbing time of the process $T(x) = \min\{t : X_t = 0$ or $X_t = m\}$ is finite and obeys:

$$\mathbb{E}\{T(x)\} \leq \frac{q(m-q)}{v} \leq \frac{m^2}{4v}$$

where $q = \mathbb{E}\{X_0\} = M(x)$.

Sketch. A similar lemma was formulated for studying asynchronous CA [6]. The main elements of its proof are: (1) to note that T is a stopping time, (2) to use the Optional Stopping Time theorem to calculate $\mathbb{E}\{X_T\}$, (3) to note that the process $Y_t = X_t^2 - v \cdot t$ is a submartingale and use again the Optional Stopping Time theorem. ◀

Proof of Theorem 1. We simply take $X_t = |x^t|_1$ and show that Lemma 2 applies to X_t. We write:

$$\begin{aligned}\mathbb{E}\{\, \Delta X_{t+1} \,|\mathcal{F}_t \,\} &= p.b + p.c + 2p.d - 2p.e - p.f - p.g \\ &= p.(b + d - e - f) + p.(c + d - e - g)\end{aligned}$$

Using Eq. (2), we obtain $\mathbb{E}\{\, \Delta X_{t+1} \,|\mathcal{F}_t \,\} = 0$.

Second, we assume that $X_t \in \{1, \ldots, n-1\}$. It implies that $x^t \notin \{\mathbf{0}, \mathbf{1}\}$, that is, x^t is not a fixed point. Denoting by $\tilde{A}$, $\tilde{B}$, the cells where transitions A, B, ... apply, and given that transitions B, C, D (resp. E, F, G) increase (resp. decrease) ΔX_{t+1} by 1, we write:

$$\Delta X_{t+1} = \sum_{c \in \tilde{B}, \tilde{C}} \mathcal{B}_c^t(p) + \sum_{c \in \tilde{D}} \mathcal{B}_c^t(2p) - \sum_{c \in \tilde{E}} \mathcal{B}_c^t(2p) - \sum_{c \in \tilde{F}, \tilde{G}} \mathcal{B}_c^t(p) \qquad (3)$$

where $(\mathcal{B}_c^t)$ is the series of the Bernoulli random variables of Eq. (1). Using the independence of these variables and $\text{var}\{\, \mathcal{B}(p) \,\} = p(1-p)$, Eq. (3) gives:

$$\begin{aligned}\text{var}\{\, \Delta X_{t+1} \,\} &= (b + c + f + g) \cdot p(1-p) + (d + e) \cdot 2p(1-2p) \\ &= p \cdot [(s_1 + 2s_2) - (s_1 + 4s_2) \cdot p]\end{aligned}$$

with $s_1 = b + c + f + g$ and $s_2 = d + e$. Using Eq. (2) and noting that the value of n is odd, we remark that there exists a 00 or 11 pattern and that $s_1 = b + c + f + g \geq 2$. From $p \leq 1/2$, we obtain $(s_1 + 2s_2) - (s_1 + 4s_2) \cdot p \geq 1$ and $\text{var}\{\, \Delta X_{t+1} \,\} \geq p$.

Lemma 2 thus applies by taking $v = p$ and $m = n$. Finally, we find that the probability that the process stops on $X_T = n$, that is, on the fixed point $\mathbf{1}$, is equal to the initial density $\rho(x) = |x|_1/n$. We also find that :

$$\forall x \in \mathcal{E}_n,\ T(x) \leq \frac{|x|_1\,(n - |x|_1)}{p} \quad \text{and} \quad T_{\text{b}} \leq \frac{n^2}{4p}$$

◀

From this result, we derive that the probability of good classification of any configuration x is equal to $\mathsf{G}(x) = \max\{\rho(x), 1 - \rho(x)\}$. The d-uniform quality of $\mathbf{C}_1$ is thus equal to $Q_{\text{d}}(n) = 3/4$ (obtained y a simple integration). For $n = 2k + 1$, the binomial quality of $\mathbf{C}_1$ is equal to: $Q_{\text{b}}(n) = 1/2 + \binom{2k}{k}/2^{2k+1}$. This formula explains why the quality of classification of $\mathbf{C}_1$ quickly decreases as the ring size n increases. For instance for $n = 49$, we have: $Q_{\text{b}}(n) = 0.557$, that is, the gain of using $\mathbf{C}_1$ compared to a random guess is less than 6%. For the reference value $n = 149$, the gain drops down to 3.3% (see Tab. 2 p. 294).

3 Schüle Density Classifier

We now consider the probabilistic density classifier proposed by Schüle et al [15]. It was designed to improve the convergence of the system towards a fixed point. For $\varepsilon \in (0, 1]$, the local rule $\mathbf{C}_2$ is defined with the following transitions:

(x, y, z)	000	001	010	011	100	101	110	111
$f(x, y, z)$	0	$1-\varepsilon$	$1-\varepsilon$	ε	$1-\varepsilon$	ε	ε	1

This rule is a density classifier as $\mathbf{0}$ and $\mathbf{1}$ are its only fixed points. With the transition code of Sec. 1.3, we write:

$$\begin{aligned}\mathbf{C}_2 &= [0, 1-\varepsilon, 1-\varepsilon, \varepsilon, \varepsilon, 1-\varepsilon, 1-\varepsilon, 0]_{\mathrm{T}} \\ &= (1-\varepsilon)\cdot \mathsf{BCFG} + \varepsilon \cdot \mathsf{DE}\end{aligned}$$

where rule $\mathsf{BCFG}(\mathbf{150})$ is the rule that implements a XOR function with three neighbours and DE is the majority rule. This means that Schüle's rule can be interpreted as applying for each cell independently: (a) a XOR with probability $1-\varepsilon$ (b) a majority with probability ε (see Fig. 1). This rule is invariant under both the left-right and the `0`-`1` symmetries (as we have: $p_{\mathrm{B}} = p_{\mathrm{C}} = p_{\mathrm{F}} = p_{\mathrm{G}}$, $p_{\mathrm{A}} = p_{\mathrm{H}}$ and $p_{\mathrm{D}} = p_{\mathrm{E}}$).

▶ **Theorem 3.** *For the classifier $\mathbf{C}_2$, for $\varepsilon = 2/3$,*

$$\forall x \in \mathcal{E}_n,\ \mathsf{G}(x) = \max\ \{\rho(x), 1-\rho(x)\} \quad \textit{and} \quad T_{\mathrm{b}} \leq 9/2 \cdot n^2$$

The relationship on $\mathsf{G}(x)$ was proved under the mean-field approximation [15]. We now propose to re-derive this result more directly.

Proof. Let us take $X_t = |x^t|_1$ and show that Lemma 2 applies to X_t. We have:

$$\begin{aligned}\mathbb{E}\{\,\Delta X_{t+1}\,\} &= (1-\varepsilon)(b+c-f-g) + \varepsilon(d-e) \\ &= (1-\varepsilon)\cdot(b+c-d+e-f-g) + d - e\end{aligned}$$

Using Eq. (2), we obtain:

$$\mathbb{E}\{\,\Delta X_{t+1}\,\} = (3\varepsilon - 2)(d-e) \tag{4}$$

which leads to $\mathbb{E}\{\,\Delta X_{t+1}\,\} = 0$ for $\varepsilon = 2/3$.

Let us now assume that $X_t \in \{1, \ldots, n-1\}$. This implies that x^t is not a fixed point and that $b+c+d+e+f+g \geq 1$. Recall that we denote by $\tilde{A}$, $\tilde{B}$,... the cells where transitions A, B ,... apply. We have:

$$\Delta X_{t+1} = \sum_{c\in\tilde{B},\tilde{C}} \mathcal{B}^t_c(1-\varepsilon) + \sum_{c\in\tilde{D}} \mathcal{B}^t_c(\varepsilon) - \sum_{c\in\tilde{E}} \mathcal{B}^t_c(\varepsilon) - \sum_{c\in\tilde{F},\tilde{G}} \mathcal{B}^t_c(1-\varepsilon)$$

where $(\mathcal{B}^t_c)$ is the series of Bernoulli random variables of Eq. (1). This results in:

$$\begin{aligned}\mathrm{var}\{\,\Delta X_{t+1}\,\} &= (b+c+d+e+f+g)\cdot\varepsilon(1-\varepsilon) \\ &\geq \varepsilon(1-\varepsilon)\end{aligned}$$

Lemma 2 thus applies by taking $v = \varepsilon(1-\varepsilon)$ and $m = n$. Consequently, we find that the probability that the process stops on the fixed point $\mathbf{1}$ (given by $X_T = n$) is equal to $X_0/n = \rho(x)$ and that $\forall x \in \mathcal{E}_n$, $T(x) \leq \frac{|x|_1(n-|x|_1)}{\epsilon(1-\epsilon)}$, which implies

$$T_{\mathrm{b}} \leq \frac{n^2}{4\epsilon(1-\epsilon)} \leq \frac{9n^2}{2}$$

◀

Equation (4) also allows us to understand the general behaviour of Schüle's classifier $\mathbf{C}_2$ for $\varepsilon \neq \frac{2}{3}$. Informally, let us consider a configuration x with a density close to 1. For such a configuration, we most likely have more isolated `0`s than isolated `1`s, that is, $d - e > 0$ and the sign of ΔX_{t+1} is the same as $3\epsilon - 2$. As for such configurations, we want the density to *increase*, we see that setting $\epsilon > 2/3$ drives the system more rapidly towards its goal. This also explains why for $\epsilon < 2/3$, it was no longer possible to observe the system convergence within "reasonable" simulation times. In fact, as observed by Schüle and al. [15], the system is then in a metastable state: although the classification time is finite, the system is always attracted towards a density 1/2. Last, but not least, we think that for $\epsilon > 2/3$, only isolated `0`s or `1`s of the *initial* configuration contribute to driving the system to its goal. This leads us to formulate the following statement:

▶ Proposition 2. For the classifier $\mathbf{C}_2$ set with $\varepsilon > 2/3$, the quality of classification $Q_{\mathrm{b}}(n)$ is bounded. More precisely:

$$\forall \epsilon > 2/3, \forall x \in \mathcal{E}_n : |x|_{\mathtt{010}} = |x|_{\mathtt{101}} = 0, \mathsf{G}(x) = \max\{\rho(x), 1 - \rho(x)\}$$

and

$$\forall \epsilon > 2/3, \forall x \in \mathcal{E}_n, \mathsf{G}(x) \leq \max\{\rho^*(x), 1 - \rho^*(x)\}$$

where $\rho^*(x) = (\Phi^{\infty}_{\mathrm{MAJ}}(x))$ is the density attained by an asymptotic evolution of x under the majority rule.

Theses hypotheses are partially confirmed by numerical simulations (see Tab. 2). We also verified experimentally that for $\epsilon \to 1$, the quality approaches an asymptotic limit while the average classification time diverges. We leave a rigorous proof this statement for future work and now present a rule that does not suffer from such limitations.

4 A New Rule for Density Classification

For $\eta \in (0, 1]$, let us consider the following sECA:

(x, y, z)	`000`	`001`	`010`	`011`	`100`	`101`	`110`	`111`
$f(x, y, z)$	0	0	0	1	$1 - \eta$	1	η	1

With the transition code, this writes:

$$\begin{aligned} \mathbf{C}_3 &= [0, 0, 1 - \eta, 1, 1, 0, 1 - \eta, 0]_{\mathrm{T}} \\ &= \eta.\mathsf{DE} + (1 - \eta).\mathsf{CDEG} \end{aligned}$$

For $\eta = 0$ we have CDEG(**184**), which is a well-known rule, often called the "traffic" rule. This rule is number conserving, *i.e.*, the number of `1`s is conserved as the system evolves (see *e.g.*, [2]). Observing the evolution of the rule, we see that a `1` with a `0` at its right moves to right while a `0` with a `1` at its left is moved to the left. So everything happens as if the `1`s were cars that tried to go to the right, possibly meeting traffic jams. These jams resorb by going in the inverse directions of the cars (when possible). For $\eta = 1$, we have the majority rule DE(**232**). For $\eta \in (0, 1)$, the effect of the rule is the same as applying, for each cell and at each time step, the traffic rule with probability $1 - \eta$ and the majority rule with probability η (see Fig. 1). This combination of rules has a surprising property: although the system is stochastic, there exists an infinity of configurations that can be classified with no error.

▶ **Lemma 4.** *An archipelago is well-classified with probability 1.*

Proof. The proof is simple and relies on two observations.

First, let us note that the successor of a q-archipelago is a q-archipelago. To see why this holds, without loss of generality, let us assume that x is a $\mathtt{1}$-archipelago. Let us denote by y a potential successor of x. Let C be the $\mathtt{1}$-cells in y: $C = \{c \in \mathcal{L} : y_c = \mathtt{1}\}$. If we look in x at the local predecessor pattern of a cell $c \in C$, we have $(x_{c-1}, x_c, x_{c+1}) \in \{\mathtt{100}, \mathtt{101}, \mathtt{011}, \mathtt{111}\}$ by examining the transition function of $\mathbf{C}_3$, and, as x is a $\mathtt{1}$-archipelago, $(x_{c-1}, x_c, x_{c+1}) \in \{\mathtt{100}, \mathtt{101}\}$. As these two patterns do not overlap, it is not possible to have two successive cells of $\mathcal{L}$ contained in C and y is a $\mathtt{1}$-archipelago.

Second, we remark that the number of $\mathtt{1}$s of x^t is a non-increasing function of t. At each time step, each isolated $\mathtt{1}$ can "disappear" if transition C is not applied, which happens with probability $\eta > 0$. As a result, all the $\mathtt{1}$s will eventually disappear and the system will attain the fixed point $\mathbf{0}$, which corresponds to a good classification as we have $\rho(x) < 1/2$. ◀

The second interesting property of $\mathbf{C}_3$ is its ability to make any configuration evolve to an archipelago with a probability that can be made as large as wanted.

▶ **Lemma 5.** *For every $p \in [0, 1)$, there exists a setting η of the classifier $\mathbf{C}_3$ such that for every configuration $x \in \mathcal{E}_n$, the probability to evolve to an archipelago x_A such that $d(x_A) = d(x)$ is greater than p.*

Proof. The proof relies on the well-known property of the traffic rule to evolve to an archipelago in at most $n/2$ steps. Let us denote by Φ the global transition function of CDEG and write $y^t = \Phi^t(x)$, that is, (y^t) is the series of configurations obtained with x as an initial condition. From the properties of the traffic rule, we have that $\rho(y^t) = \rho(x)$ and that $y^{\lceil n/2 \rceil}$ is an archipelago[4] (see *e.g.*, Ref. [3] Lemma 4).

For a given p and given n, without loss of generality, let us consider a configuration x such that $\rho(x) < 1/2$. Let us now evaluate the probability that rule $\mathbf{C}_3$ does *not* behave like the traffic rule in the first $T = \lceil n/2 \rceil$ steps. Formally let $D^t = card\{c \in \mathcal{L} : x^t_c \neq y^t_c\}$.

Comparing the transition rules of CDEG and $\mathbf{C}_3$, we see that differences in the evolution of the two rules can only occur for cells where transitions C and G apply, that is, cells that have a $\mathtt{100}$ and $\mathtt{110}$ neighbourhood. As we have $b = c$ and $f = g$, and $b + f + g + c \leq n$, we write $b + f \leq \lceil n/2 \rceil$. For such cells, differences of evolution occur with a probability η, which implies that, at each time step, the probability $p_{\text{diff}} = \Pr\{D^t > 0\}$ that the evolution of $\mathbf{C}_3$ and CDEG differ on T steps is upper-bounded by: $p_{\text{diff}} \leq \eta^{T \cdot \lceil n/2 \rceil} \leq \eta^{T^2}$. The probability $P_{\text{eq}} = \Pr\{D^1 = 0, \ldots, D^T = 0\}$ that the two rules evolve identically on T steps is thus greater than or equal to $1 - p_{\text{diff}}$ and we find that it is sufficient to set: $\eta < 1 - p^{\frac{1}{T^2}}$ to guarantee that $P_{\text{eq}} > p$, *i.e.*, that the probability to reach a $\mathtt{1}$-archipelago is greater than p. As the traffic rule is number-conserving, the archipelago has the same density as the initial configuration. ◀

This inequality shows that, by taking η small enough, the probability that a configuration x with $\rho(x) < 1/2$ evolves to a $\mathtt{1}$-archipelago can be made arbitrarily small. This allows us to state our main result.

▶ **Theorem 6.** *For all $p \in [0, 1)$, there exists a setting η of the classifier $\mathbf{C}_3$ such that $\forall x \in \mathcal{E}_n$, $G(x) \geq p$. As a consequence, $\forall n \in 2\mathbb{N} + 1$, setting $\eta \to 0$ implies $Q_{\text{b}}(n) \to 1$.*

[4] As remarked by an anonymous referee, $\lfloor n/2 \rfloor$ steps should be sufficient.

Table 2 Results for $n = 149$; averages on 10 000 samples, the values 53.3 and 75.0 are calculated.

model	setting	Q_b (in%)	Q_d (in%)	T_b
$\mathbf{C}_1$	$p = 0.25$	53.3	75.0	4638
$\mathbf{C}_1$	$p = 0.48$	53.3	75.0	2652
$\mathbf{C}_1$	$p = 0.5$	53.3	75.0	8985
$\mathbf{C}_2$	$\epsilon = 0.7$	54.0	80.1	4061
$\mathbf{C}_2$	$\epsilon = 0.8$	55.1	83.8	6223
$\mathbf{C}_2$	$\epsilon = 0.9$	56.6	85.8	11887
$\mathbf{C}_3$	$\eta = 0.1$	82.4	98.1	517
$\mathbf{C}_3$	$\eta = 0.01$	91.0	99.1	4950
$\mathbf{C}_3$	$\eta = 0.005$	93.4	99.3	9981

Proof. Combining the two previous lemmas to prove the theorem is straightforward: for η small enough, the system evolves to an archipelago that has the same density as the initial condition (Lemma 5). It is then necessarily well-classified as it will progressively "drift" towards the appropriate fixed point (Lemma 4). However, we remark that the time taken to reach the fixed point increases as η decreases. ◀

The analytical estimation of the quality of $\mathbf{C}_3$ and its time of convergence is more complex than for Fukś and Schüle classifiers. Table 2 shows the values of Q_b, Q_d and T_b estimated by numerical simulations. We can observe that the quality rapidly increases to high values, even when keeping the average convergence time to a few thousand steps. In particular for $\eta < 1\%$, the quality goes above the symbolic rate of 90%, which, to our knowledge, has not been yet reached for one-dimensional systems (see *e.g.* [4, 12]). Another major point regards the classification time of $\mathbf{C}_3$: for $n \leq 300$ and $\eta \leq 0.1$, it is experimentally determined as varying linearly (or quasi-linearly) with the ring size n.

References

1 Ramón Alonso-Sanz and Larry Bull. A very effective density classifier two-dimensional cellular automaton with memory. *Journal of Physics A*, 42(48):485101, 2009.

2 Nino Boccara and Henryk Fukś. Number-conserving cellular automaton rules. *Fundamenta Informaticae*, 52(1-3):1–13, 2002.

3 Mathieu S. Capcarrere, Moshe Sipper, and Marco Tomassini. Two-state, $r = 1$ cellular automaton that classifies density. *Phys. Rev. Lett.*, 77(24):4969–4971, 1996.

4 Pedro P.B. de Oliveira, José C. Bortot, and Gina M.B. Oliveira. The best currently known class of dynamically equivalent cellular automata rules for density classification. *Neurocomputing*, 70(1-3):35 – 43, 2006.

5 Paula Gonzaga de Sá and Christian Maes. The Gacs-Kurdyumov-Levin automaton revisited. *Journal of Statistical Physics*, 67:507–522, 1992.

6 Nazim Fatès, Michel Morvan, Nicolas Schabanel, and Eric Thierry. Fully asynchronous behavior of double-quiescent elementary cellular automata. *Theoretical Computer Science*, 362:1–16, 2006.

7 Henryk Fukś. Solution of the density classification problem with two cellular automata rules. *Physical Review E*, 55(3):R2081–R2084, Mar 1997.

8 Henryk Fukś. Nondeterministic density classification with diffusive probabilistic cellular automata. *Physical Review E*, 66(6):066106, 2002.

9 Mark Land and Richard K. Belew. No perfect two-state cellular automata for density classification exists. *Physical Review Letters*, 74(25):5148–5150, 1995.

10 Claudio L.M. Martins and Pedro P.B. de Oliveira. Evolving sequential combinations of elementary cellular automata rules. In Mathieu S. Capcarrere, Alex A. Freitas, Peter J. Bentley, Colin G. Johnson, and Jon Timmis, editors, *Advances in Artificial Life*, volume 3630 of *Lecture Notes in Computer Science*, pages 461–470. Springer Berlin Heidelberg, 2005.

11 Melanie Mitchell, James P. Crutchfield, and Peter T. Hraber. Evolving cellular automata to perform computations: Mechanisms and impediments. *Physica D*, 75:361–391, 1994.

12 Gina M. B. Oliveira, Luiz G. A. Martins, Laura B. de Carvalho, and Enrique Fynn. Some investigations about synchronization and density classification tasks in one-dimensional and two-dimensional cellular automata rule spaces. *Electronic Notes in Theoretical Computer Science*, 252:121–142, 2009.

13 Norman H. Packard. *Dynamic Patterns in Complex Systems*, chapter Adaptation toward the edge of chaos, pages 293 – 301. World Scientific, Singapore, 1988.

14 Leonid A. Levin Peter Gács, Georgii L. Kurdiumov. One-dimensional homogeneous media dissolving finite islands. *Problemy Peredachi Informatsii*, 14:92–96, 1987.

15 Martin Schüle, Thomas Ott, and Ruedi Stoop. Computing with probabilistic cellular automata. In *ICANN '09: Proceedings of the 19th International Conference on Artificial Neural Networks*, pages 525–533, Berlin, Heidelberg, 2009. Springer-Verlag.

16 Christopher Stone and Larry Bull. Evolution of cellular automata with memory: The density classification task. *BioSystems*, 97(2):108–116, 2009.

Probabilistic cellular automata, invariant measures, and perfect sampling

Ana Bušić[1], Jean Mairesse[2], and Irène Marcovici[2,3]

1 **INRIA/ENS**
23, avenue d'Italie, CS 81321, 75214 Paris Cedex 13, France
`Ana.Busic@inria.fr`

2 **LIAFA, CNRS and Université Paris Diderot - Paris 7**
Case 7014, 75205 Paris Cedex 13, France
`(Jean.Mairesse,Irene.Marcovici)@liafa.jussieu.fr`

3 **ENS Lyon, Département de Mathématiques**

Abstract

In a probabilistic cellular automaton (PCA), the cells are updated synchronously and independently, according to a distribution depending on a finite neighborhood. A PCA can be viewed as a Markov chain whose ergodicity is investigated. A classical cellular automaton (CA) is a particular case of PCA. For a 1-dimensional CA, we prove that ergodicity is equivalent to nilpotency, and is therefore undecidable. We then propose an efficient perfect sampling algorithm for the invariant measure of an ergodic PCA. Our algorithm does not assume any monotonicity property of the local rule. It is based on a bounding process which is shown to be also a PCA.

1998 ACM Subject Classification G.3; F.1.2

Keywords and phrases probabilistic cellular automata, perfect sampling, ergodicity.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.296

1 Introduction

Cellular automata (CA) are dynamical systems in which space and time are discrete. A cellular automaton consists of a lattice (e.g. $\mathbb{Z}^d$ or $\mathbb{Z}/n\mathbb{Z}$) divided in regular cells, each cell containing a letter of a finite alphabet. The cells evolve synchronously, each one evolving in function of a finite number of cells in its neighborhood, according to a local rule.

To take into account randomness, one is led to consider *probabilistic cellular automata* (PCA) [17]. For PCA, time is dicrete and the cells evolve synchronously as for CA, but the difference is that for each cell, the new content is randomly chosen, independently of the others, according to a distribution depending only on a finite neighborhood of the cell.

Let us mention a couple of motivations. First, the investigation of fault-tolerant computational models was the motivation for the russian school to study PCA [17, 6]. Second, PCA appear in combinatorial problems related to the enumeration of directed animals [11]. Third, in the context of the classication of CA (Wolfram's program), robustness to random errors can be used as a discriminating criterion [5, 14].

We focus our study on the equilibrium behavior of PCA. Observe that a PCA may be viewed as a Markov chain over the state space $\mathcal{A}^E$, where $\mathcal{A}$ is the alphabet and E is the set of cells. The equilibrium is studied via the invariant measures of the Markov chain. A PCA is *ergodic* if it has a unique and attractive invariant measure. Finding conditions to ensure ergodicity is a difficult problem which has been thoroughly investigated [17, 6]. When a PCA is ergodic, it is usually impossible to determine the invariant measure explicitly, and

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 296–307
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

simulation becomes the alternative. Simulating PCA is known to be a challenging task, costly both in time and space. Also, configurations cannot be tracked down one by one (there is an infinite number of them when E is infinite) and may only be observed through some measured parameters. The point is to have guarantees upon the results obtained from simulations.

In this context, our contributions are as follows. First, we prove that the ergodicity of a CA on $\mathbb{Z}$ is undecidable. This was mentioned as *Unsolved Problem 4.5* in [16]. Since a CA is a special case of a PCA, it also provides a new proof of the undecidability of the ergodicity of a PCA (Kurdyumov, see [17, Chap. 14], and Toom [15]). Second, we propose an efficient perfect sampling algorithm for ergodic PCA. Recall that a *perfect sampling* procedure is a random algorithm which returns a configuration distributed according to the invariant measure. By applying the procedure repeatedly, we can estimate the invariant measure with arbitrary precision. We propose such an algorithm for PCA by adapting the *coupling from the past* method of Propp & Wilson [12]. When the set of cells is finite, a PCA is a finite state space Markov chain. Therefore, coupling from the past from all possible initial configurations provides a basic perfect sampling procedure, but a very inefficient one since the number of configurations is exponential in the number of cells. Here, the contribution consists in an important simplification of the procedure. We define a new PCA on an extended alphabet, called the *envelope PCA* (EPCA). We obtain a perfect sampling procedure for the original PCA by running the EPCA on a single initial configuration. When the set of cells is infinite, a PCA is a Markov chain on an uncountable state space. So there is no basic perfect sampling procedure anymore. We prove the following: If the PCA is ergodic, then the EPCA may or may not be ergodic. If it is ergodic, then we can use the EPCA to design an efficient perfect sampling procedure (the result of the algorithm is the finite restriction of a configuration with the right invariant distribution). The EPCA can be viewed as a systematic treatment of ideas already used by Toom for *percolation PCA* (see for instance [16, Section 2]).

The perfect sampling procedure can also be run on a PCA whose ergodicity is unknown, with the purpose of testing it. We illustrate this approach on *Majority*, prototype of a PCA whose equilibrium behavior is not well understood.

2 Probabilistic cellular automata

Let $\mathcal{A}$ be a finite set called the *alphabet*, and let E be a countable or finite set of *cells*. We denote by X the set $\mathcal{A}^E$ of *configurations*.

We assume that E is equipped with a commutative semigroup structure, whose law is denoted by $+$. In examples, we consider mostly the cases $E = \mathbb{Z}$ or $E = \mathbb{Z}/n\mathbb{Z}$. Given $K \subset E$ and $V \subset E$, we define $V + K = \{v + k \mid v \in V, k \in K\}$.

A *cylinder* is a subset of X having the form $\{x \in X \mid \forall k \in K, x_k = y_k\}$ for a given finite subset K of E and a given element $(y_k)_{k \in K} \in \mathcal{A}^K$. When there is no possible confusion, we shall denote briefly by y_K the cylinder $\{x \in X \mid \forall k \in K, x_k = y_k\}$. For a given finite subset K, we denote by $\mathcal{C}(K)$ the set of all cylinders of base K.

Let us equip $X = \mathcal{A}^E$ with the product topology, which can be described as the topology generated by cylinders. We denote by $\mathcal{M}(\mathcal{A})$ the set of probability measures on $\mathcal{A}$ and by $\mathcal{M}(X)$ the set of probability measures on X for the σ-algebra generated by all cylinder sets, which corresponds to the Borelian σ-algebra. For $x \in X$, denote by δ_x the Dirac measure concentrated on the configuration x.

▶ **Definition 2.1.** Given a finite set $V \subset E$, a *transition function* of *neighborhood* V is a function $f : \mathcal{A}^V \to \mathcal{M}(\mathcal{A})$. The *probabilistic cellular automaton* (PCA) P of transition function f is the application $P : \mathcal{M}(X) \to \mathcal{M}(X),\ \mu \mapsto \mu P$, defined on cylinders by:

$$\mu P(y_K) = \sum_{x_{V+K} \in \mathcal{C}(V+K)} \mu(x_{V+K}) \prod_{k \in K} f((x_{k+v})_{v\in V})(y_k)\,.$$

Let us look at how P acts on a Dirac measure δ_z. The content z_k of the k-th cell is changed into the letter $a \in \mathcal{A}$ with probability $f((z_{k+v})_{v\in V})(a)$, independently of the evolution of the other cells. The real number $f((z_{k+v})_{v\in V})(a) \in [0,1]$ is thus to be thought as the conditional probability that, after application of P, the k-th cell will be in the state a if, before its application, the neighborhood of k was in the state $(z_{k+v})_{v\in V}$.

Let u be the uniform measure on $[0,1]$. We define the product measure $\tau = \bigotimes_{i\in E} u$ on $[0,1]^E$.

▶ **Definition 2.2.** An *update function* of the probabilistic cellular automaton P is a deterministic function $\phi : \mathcal{A}^E \times [0,1]^E \to \mathcal{A}^E$ (the function ϕ takes as argument a configuration and a sample in $[0,1]^E$, and returns a new configuration), satisfying for each $x \in \mathcal{A}^E$, and each cylinder y_K,

$$\tau(\{r \in [0,1]^E ; \phi(x,r) \in y_K\}) = \prod_{k\in K} f((x_{k+v})_{v\in V})(y_k).$$

In practice, it is always possible to define an update function ϕ for which the value of $\phi(x,r)_k$ only depends on $(x_{k+v})_{v\in V}$ and on r_k. For example, if the alphabet is $\mathcal{A} = \{a_1,\ldots,a_n\}$, one can set

$$\phi(x,r)_k = \begin{cases} a_1 & \text{if } \ 0 \le r_k < f((x_{k+v})_{v\in V})(a_1) \\ a_2 & \text{if } \ f((x_{k+v})_{v\in V})(a_1) \le r_k < f((x_{k+v})_{v\in V})(\{a_1,a_2\}) \\ \vdots & \\ a_n & \text{if } \ f((x_{k+v})_{v\in V}(\{a_1,a_2,\ldots,a_{n-1}\}) \le r_k \le 1. \end{cases} \tag{1}$$

For a given initial configuration $x^0 \in \mathcal{A}^E$, and samples $(r^t)_{t\in\mathbb{N}}$, $r^t \in [0,1]^E$, let $(x^t)_{t\in\mathbb{N}} \in (\mathcal{A}^E)^{\mathbb{N}}$ be the sequence defined recursively by $x^{t+1} = \phi(x^t, r^t)$. Such a sequence is called a *space-time diagram.* It can be viewed as a realization of the Markov chain. Examples of space-time diagrams appear in Figures 1 and 2.

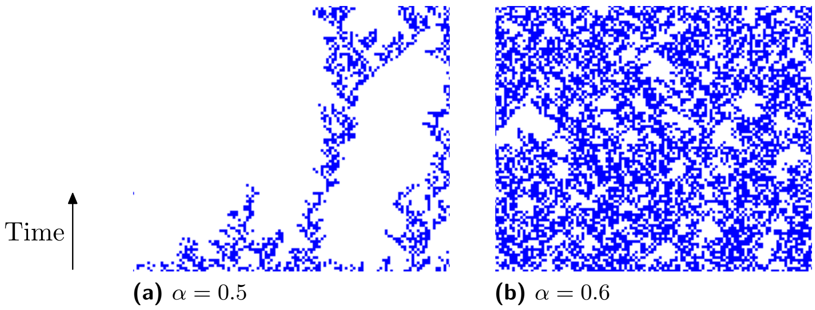
Classical cellular automata are a specialization of PCA.

▶ **Definition 2.3.** A *deterministic cellular automaton* (DCA) is a PCA such that for each sequence $(x_v)_{v\in V} \in \mathcal{A}^V$, the measure $f((x_v)_{v\in V})$ is concentrated on a single letter of the alphabet. A DCA can thus be seen as a deterministic function $F : \mathcal{A}^E \to \mathcal{A}^E$.

In the literature, the term *cellular automaton* denotes what we call here a DCA. Deterministic cellular automata have been widely studied, in particular on the set of cells $E = \mathbb{Z}$, see Section 3. For a DCA, any initial configuration defines a unique space-time diagram.

▶ **Example 2.4.** Let $\mathcal{A} = \{0,1\}$, $E = \mathbb{Z}$, and $V = \{0,1\}$. Consider $0 < \varepsilon < 1$ and the local function $f(x,y) = (1-\varepsilon)\,\delta_{x+y \mod 2} + \varepsilon\,\delta_{x+y+1 \mod 2}$. This defines a PCA that can be considered as a perturbation of the DCA $F : \mathcal{A}^E \to \mathcal{A}^E$ defined by $F(x)_i = x_i + x_{i+1}$ mod 2, with errors occuring in each cell independently with probability ε.

▶ **Example 2.5.** Let $\mathcal{A} = \{0,1\}$, $E = \mathbb{Z}^d$, and let V be a finite subset of E. Consider $0 < \alpha < 1$ and the local function: $f((x_v)_{v\in V}) = \alpha\,\delta_{\max(x_v,\ v\in V)} + (1-\alpha)\,\delta_0$. The corresponding PCA is called the *percolation PCA* associated with V and α. The particular case of the space $E = \mathbb{Z}$ and the neighborhood $V = \{0,1\}$ is called the *Stavskaya PCA*. In Figure 1, we represent two portions of diagrams of the percolation PCA for $E = \mathbb{Z}$ and $V = \{-1,0,1\}$.



(a) $\alpha = 0.5$ **(b)** $\alpha = 0.6$

Figure 1 Space-time diagrams of the PCA of Example 2.5, for $V = \{-1, 0, 1\}$.

2.1 Invariant measures and ergodicity

A PCA can be seen as a Markov chain on the state space $\mathcal{A}^E$. We use the classical terminology for Markov chains that we now recall.

▶ **Definition 2.6.** A probability measure $\pi \in \mathcal{M}(X)$ is said to be an *invariant measure* of the PCA P if $\pi P = \pi$. The PCA is *ergodic* if it has exactly one invariant measure π which is *attractive*, that is, for any measure $\mu \in \mathcal{M}(X)$, the sequence μP^n converges weakly to π (*i.e.* for any cylinder C, $\lim_{n \to +\infty} \mu P^n(C) = \pi(C)$).

A PCA has at least one invariant measure, and the set of invariant measures is convex and compact. This is a standard fact, based on the observation that the set $\mathcal{M}(X)$ of measures on X is compact for the weak topology, see for instance [17]. Therefore, there are three possible situations for a PCA:

(i) several invariant measures; (ii) a unique invariant measure which is not attractive; (iii) a unique invariant measure which is attractive (ergodic case).

▶ **Example 2.7.** Consider the PCA of Example 2.4. Using the results in [17, Chapters 16 and 17], one can prove that the PCA is ergodic and that its unique invariant measure is the uniform mesure, *i.e.* the product of Bernoulli measures of parameter 1/2.

▶ **Example 2.8.** Consider the percolation PCA of Example 2.5. Observe that the Dirac measure δ_{0^E} is an invariant measure. Using a coupling with a percolation model, one can prove the following, see for instance [16, Section 2]. There exists $\alpha^* \in (0, 1)$ such that:

$$\alpha < \alpha^* \implies (iii): \text{ ergodicity}, \qquad \alpha > \alpha^* \implies (i): \text{ several invariant measures.}$$

The exact value of α^* is not known but it satisfies $1/|V| \leq \alpha^* \leq 53/54$.

The existence of a PCA corresponding to situation (ii) had been a long standing conjecture, but an example has recently been presented in [3]. The PCA of Example 2.5 exhibits a phase transition between the situations (i) and (iii). In Section 5, we study a PCA that may have a phase transition between the situations (ii) and (iii). It would provide the first example of this type.

3 Ergodicity of DCA

DCA form the simplest class of PCA, it is therefore natural to study the ergodicity of DCA. In this section, we prove the undecidability of ergodicity for DCA (Theorem 3.4).

Remark. In the context of DCA, the terminology of Definition 2.6 might be confusing. Indeed a DCA P can be viewed in two different ways: (i) a (degenerated) Markov chain;

(*ii*) a symbolic dynamical system. In the dynamical system terminology, P is *uniquely ergodic* if: $[\exists!\mu,\ \mu P = \mu]$. In the Markov chain terminology (that we adopt), P is *ergodic* if: $[\exists!\mu,\ \mu P = \mu]$ and $[\forall\nu,\ \nu P^n \xrightarrow{w} \mu]$, where $\xrightarrow{w}$ stands for the weak convergence. Knowing if the unique ergodicity (of symbolic dynamics) implies the ergodicity (of the Markov theory) is an open question for DCA.

The *limit set* of P is defined by $LS = \bigcap_{n\in\mathbb{N}} P^n(\mathcal{A}^E)$. In words, a configuration belongs to LS if it may occur after an arbitrarily long evolution of the cellular automaton. Observe that LS is non-empty since it is the decreasing limit of non-empty closed sets. A constructive way to show that LS is non-empty is as follows. The image by P of a monochromatic configuration x^E is monochromatic: $x^E \to y^E$. In particular, there exists a monochromatic periodic orbit for P, and we have: $x_0^E \to x_1^E \to \cdots \to x_{k-1}^E \to x_0^E \implies \{x_0^E, x_1^E, \ldots, x_{k-1}^E\} \subset LS$.

Recall that δ_u denotes the probability measure concentrated on the configuration u. The periodic orbit $(x_0^E, \ldots, x_{k-1}^E)$ provides an invariant measure given by $(\delta_{x_0^E} + \ldots + \delta_{x_{k-1}^E})/k$. More generally, the support of any invariant measure is included in the limit set.

▶ **Definition 3.1.** A DCA is *nilpotent* if its limit set is a singleton.

Clearly, a DCA is nilpotent iff $LS = \{x^E\}$ for some $x \in \mathcal{A}$. The following stronger statement is proved in [4], using a compactness argument:

$$[\ P \textit{ nilpotent}\ ] \quad \iff \quad [\ \exists x \in \mathcal{A}, \exists N \in \mathbb{N},\ \ P^N(\mathcal{A}^E) = \{x^E\}\ ]\ .$$

In that case, for any probability measure μ on $\mathcal{A}^E$, we have $\mu P^N = \delta_{x^E}$, so that P is ergodic with unique invariant measure δ_{x^E}. This proves the following proposition.

▶ **Proposition 3.2.** *Consider a DCA P. We have: $[\ P$ nilpotent $] \implies [\ P$ ergodic $]$.*

If we restrict ourselves to DCA on $\mathbb{Z}$, we get the converse statement.

▶ **Theorem 3.3.** *Consider a DCA P on the set of cells $\mathbb{Z}$. We have:*

$$[\ P \textit{ nilpotent}\ ] \quad \iff \quad [\ P \textit{ ergodic}\ ]\ .$$

Proof. Let P be an ergodic DCA. Assume that there exists a monochromatic periodic orbit $(x_0^{\mathbb{Z}}, \ldots, x_{k-1}^{\mathbb{Z}})$ with $k \geq 2$. Then $\mu = (\delta_{x_0^{\mathbb{Z}}} + \cdots + \delta_{x_{k-1}^{\mathbb{Z}}})/k$ is the unique invariant measure. The sequence $\delta_{x_0^{\mathbb{Z}}} P^n$ does not converge weakly to μ, which is a contradiction. Therefore, there exists a monochromatic fixed point: $P(x^{\mathbb{Z}}) = x^{\mathbb{Z}}$, and $\delta_{x^{\mathbb{Z}}}$ is the unique invariant measure.

Define the cylinder $C = \{v \in \mathcal{A}^{\mathbb{Z}} \mid \forall i \in K,\ v_i = x\}$, where K is some finite subset of $\mathbb{Z}$. For any initial configuration $u \in \mathcal{A}^{\mathbb{Z}}$, using the ergodicity of P, we have: $\delta_u P^n(C) \longrightarrow \delta_{x^{\mathbb{Z}}}(C) = 1$. But $\delta_u P^n$ is a Dirac measure, so $\delta_u P^n(C)$ is equal to 0 or 1. Consequently, we have $\delta_u P^n(C) = 1$ for n large enough, that is, $\exists N \in \mathbb{N}, \forall n \geq N, \forall i \in K, \quad P^n(u)_i = x$.

In words, in any space-time diagram of P, any column becomes eventually equal to $xxx\cdots$. Using the terminology of Guillon & Richard [8], the DCA P has a *weakly nilpotent trace*. It is proved in [8] that the weak nilpotency of the trace implies the nilpotency of the DCA. (The result is proved for cellular automata on $\mathbb{Z}$ and left open in larger dimensions.) This completes the proof. ◀

Kari proved in [10] that the nilpotency of a DCA on $\mathbb{Z}$ is undecidable. (For DCA on $\mathbb{Z}^d$, $d \geq 2$, the proof appears in [4].) By coupling Kari's result with Theorem 3.3, we get:

▶ **Corollary 3.4.** *Consider a DCA P on the set of cells $\mathbb{Z}$. The ergodicity of P in undecidable.*

The undecidability of the ergodicity of a PCA was a known result, proved by Kurdyumov, see [17], see also Toom [15]. But the undecidability of the ergodicity of a DCA, which is a stronger result, was in fact mentioned as *Unsolved Problem 4.5* in [16].

Corollary 3.4 can also be obtained without Theorem 3.3, by directly adapting Kari's proof to show the undecidability of the ergodicity of the DCA associated with a NW-deterministic tile set.

4 Sampling the invariant measure of an ergodic PCA

Generally, the invariant measure(s) of a PCA cannot be described explicitly. Numerical simulations are consequently very useful to get an idea of the behavior of a PCA. Given an ergodic PCA, we propose a *perfect sampling* algorithm which generates configurations *exactly* according to the invariant measure.

A perfect sampling procedure for finite Markov chains has been proposed by Propp & Wilson [12] using a *coupling from the past* scheme. Perfect sampling procedures have been developed since in various contexts. Let us mention some related works. For more information see the annotated bibliography: *Perfectly Random Sampling with Markov Chains*, `http://dimacs.rutgers.edu/~dbwilson/exact.html/`.

The complexity of the algorithm depends on the number of all possible initial conditions, which is prohibitive for PCA. A first crucial observation already appears in [12]: for a monotone Markov chain, one has to consider only extremal initial conditions. To cope with more general situations, Huber [9] introduced the idea of a bounding chain for determining when coupling has occurred. The construction of these bounding chains is model-dependent and in general not straightforward. In the case of a Markov chain on a lattice, Bušić et al. [2] proposed an algorithm to construct the bounding chains.

Our contribution is to show that the bounding chain ideas can be given in a particularly simple and convenient form in the context of PCA via the introduction of the *envelope PCA*.

4.1 Basic coupling from the past for PCA

We present first the algorithm for a PCA on a finite set of cells, and then for an infinite set of cells.

Finite set of cells. Consider an ergodic PCA P on the alphabet $\mathcal{A}$ and on a finite set of cells E (for example $\mathbb{Z}_m = \mathbb{Z}/m\mathbb{Z}$). Let π be the invariant measure on $X = \mathcal{A}^E$. A *perfect sampling* procedure is a random algorithm which returns a state $x \in X$ with probability $\pi(x)$. Algorithm 1 is a presentation of the Propp & Wilson, or *coupling from the past (CFTP)*, perfect sampling procedure, written here in the context of PCA.

Algorithm 1: Basic CFTP algorithm for a finite set of cells

```
Data: Update function $\phi : X \times [0,1]^E \to X$ of
      a PCA. Family $(r_k^{-n})_{(k,n)\in E\times\mathbb{N}}$ of i.i.d.
      r.v. uniform on $[0,1]$.
begin
    $t = 1$ ;
    repeat
        $R_{-t} = X$ ;
        for $j = -t$ to $-1$ do
            $R_{j+1} = \{\phi(x, (r_i^j)_{i\in E}) ; x \in R_j\}$
        $t = t + 1$
    until $|R_0| = 1$ ;
    return the unique element of $R_0$
end
```

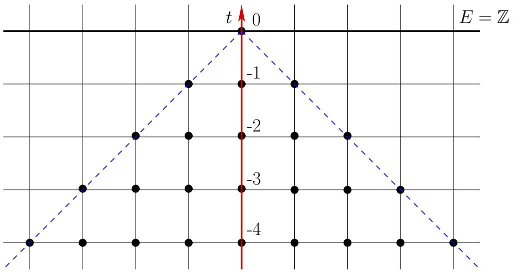
▶ **Proposition 4.1** ([12]). *If Algorithm 1 stops almost surely, then the PCA is ergodic and the output is distributed according to the invariant measure.*

The proof is based on the following idea: if we had run the Markov chain from time $-\infty$ up to 0, then the result would obviously be equal to the output of the algorithm. But if we start from time $-\infty$, the Markov chain has reached equilibrium by time 0.

Infinite set of cells. Assume that the set of cells E is infinite. Then a PCA defines a Markov chain on the infinite state space $X = \mathcal{A}^E$, so the above procedure is not effective anymore. However, it is possible to use the locality of the updating rule of a PCA to still define a perfect sampling procedure. (This observation already appears in [1].)

Let P be an ergodic PCA P and denote by π its invariant distribution. In this context, a *perfect sampling* procedure is a random algorithm taking as input a finite subset K of E and returning a cylinder $x_K \in \mathcal{C}(K)$ with probability $\pi(x_K)$.

To get such a procedure, we use the following fact: if the PCA is run from time $-k$ onwards, then to compute the content of the cells in K at time 0, it is enough to consider the cells in the finite dependence cone of K. This is illustrated here for the set of cells $E = \mathbb{Z}$ and the neighborhood $V = \{-1, 0, 1\}$, with the choice $K = \{0\}$.



More formally, let V be the neighborhood of the PCA. Given a subset K of E, the *dependence cone* of K is the family $(V_{-t}(K))_{t\in\mathbb{N}}$ of subsets of E defined recursively by $V_0(K) = K$ and $V_{-t}(K) = V + V_{-t+1}(K)$. Let $\phi : X \times [0,1]^E \to X$ be an update function, for instance the one defined in (1). For a given subset K of E, we denote $\phi_{-t} : \mathcal{A}^{V_{-t}(K)} \times [0,1]^{V_{-t}(K)} \to \mathcal{A}^{V_{-t+1}(K)}$ the corresponding restriction of ϕ. With these notations, the algorithm now consists in setting at each step $R_{-t} = \mathcal{A}^{V_{-t}(K)}$ and computing $R_{j+1} = \{\phi_j(x, (r_i^j)_{i\in V_j(K)}) \, ; \, x \in R_j\} \subset \mathcal{A}^{V_{j+1}(K)}$ for $j = -t$ to -1. This is done until we get $|R_0| = 1$.

Next proposition is an easy extension of Proposition 4.1.

▶ **Proposition 4.2.** *If the procedure stops almost surely, then the PCA is ergodic and the output is distributed according to the marginal of the invariant measure.*

4.2 Envelope probabilistic cellular automata (EPCA)

The CFTP algorithm is inefficient when the state space is large. This is the case for PCA: when E is finite, the set $\mathcal{A}^E$ is very large, and when E is infinite, it is the dependence cone described above which is very large. We cope with this difficulty by introducing the *envelope* PCA.

For simplicity, we assume that P is a PCA on the alphabet $\mathcal{A} = \{0, 1\}$ (as previously, the set of cells is denoted by E, the neighborhood by $V \subset E$ and the local function by f). Most of the results can be easily extended to the case of a general alphabet.

Definition of the EPCA. Let us introduce a new alphabet: $\mathcal{B} = \{\mathbf{0}, \mathbf{1}, ?\}$. A word on $\mathcal{B}$ is to be thought as a word on $\mathcal{A}$ in which the letters corresponding to some positions are not known, and are thus replaced by the symbol "?". Formally we identify $\mathcal{B}$ with $2^{\mathcal{A}} - \emptyset$ as follows: $\mathbf{0} = \{0\}$, $\mathbf{1} = \{1\}$, and $? = \{0, 1\}$. So each letter of $\mathcal{B}$ is a set of possible letters of $\mathcal{A}$. With this interpretation, we view a word on $\mathcal{B}$ as a set of words on $\mathcal{A}$. For instance, $?\mathbf{1}? = \{010, 011, 110, 111\}$.

We will associate to the PCA P a new PCA on the alphabet $\mathcal{B}$, that we call the *envelope probabilistic cellular automaton* of P.

▶ **Definition 4.3.** The *envelope probabilistic cellular automaton (EPCA)* of P, is the PCA $\mathrm{env}(P)$ of alphabet $\mathcal{B}$, defined on the set of cells E, with the same neighborhood V as for P, and a local function $\mathrm{env}(f) : \mathcal{B}^V \to \mathcal{M}(\mathcal{B})$ defined for each $y \in \mathcal{B}^V$ by

$$\mathrm{env}(f)(y)(\mathbf{0}) = \min_{x\in\mathcal{A}^V,\, x\in y} f(x)(0), \qquad \mathrm{env}(f)(y)(\mathbf{1}) = \min_{x\in\mathcal{A}^V,\, x\in y} f(x)(1)$$

$$\mathrm{env}(f)(y)(?) = 1 - \min_{x\in\mathcal{A}^V,\, x\in y} f(x)(0) - \min_{x\in\mathcal{A}^V,\, x\in y} f(x)(1).$$

Observe that $\mathrm{env}(P)$ acts like P on configurations which do not contain the letter “?”. More precisely,

$$\forall y \in \mathcal{A}^V, \qquad \mathrm{env}(f)(y)(\mathbf{0}) = f(y)(0),\ \mathrm{env}(f)(y)(\mathbf{1}) = f(y)(1),\ \mathrm{env}(f)(y)(?) = 0\,. \tag{2}$$

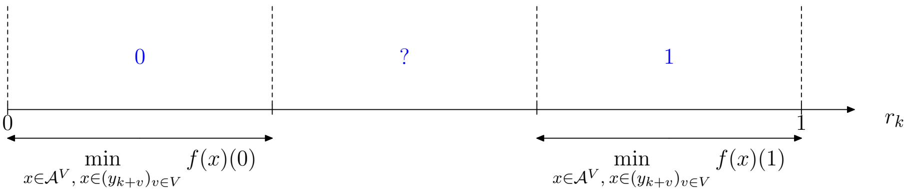
It implies next proposition. The converse statement is not true, see the counter-examples in Section 4.3.3.

▶ **Proposition 4.4.** *If the EPCA $\mathrm{env}(P)$ is ergodic then the PCA P is ergodic.*

Construction of an update function for the EPCA. Let us define the update function $\tilde{\phi} : \mathcal{B}^E \times [0,1]^E \to \mathcal{B}^E$ of the PCA $\mathrm{env}(P)$, by:

$$\tilde{\phi}(y,r)_k = \begin{cases} \mathbf{0} \text{ if } 0 \le r_k < \mathrm{env}(f)((y_{k+v})_{v\in V})(\mathbf{0}) \\ \mathbf{1} \text{ if } 1 - \mathrm{env}(f)((y_{k+v})_{v\in V})(\mathbf{1}) \le r_k \le 1 \\ ? \text{ otherwise.} \end{cases} \tag{3}$$

The value of $\tilde{\phi}(y,r)_k$ in function of r_k can be represented as follows:


0
?
1
0
1
r_k
$\min_{x\in\mathcal{A}^V,\, x\in(y_{k+v})_{v\in V}} f(x)(0)$
$\min_{x\in\mathcal{A}^V,\, x\in(y_{k+v})_{v\in V}} f(x)(1)$

Let ϕ be the natural update function for the PCA P defined in (1). Observe that $\tilde{\phi}$ coincides with ϕ on configurations which do not contain the letter “?”. Furthermore, we have:

$$\forall r \in [0,1]^E,\ \forall x \in \mathcal{A}^E,\ \forall y \in \mathcal{B}^E, \qquad x \in y \implies \phi(x,r) \in \tilde{\phi}(y,r)\,. \tag{4}$$

4.3 Perfect sampling using EPCA

We propose two perfect sampling algorithms, for a finite and for an infinite number of cells. We show that in both cases, the algorithm stops almost surely if and only if the EPCA is ergodic. The ergodicity of the EPCA implies the ergodicity of the PCA but the converse is not true: we provide a counterexample for each case, finite and infinite.

We also give sufficient conditions of ergodicity of the EPCA.

4.3.1 Algorithms

The algorithm for a finite set of cells is given in Algorithm 2. For an infinite set of cells, we consider the dependence cone of a finite set of cells K (see Section 4.1).

Finite set of cells. The idea is to consider only one trajectory of the EPCA - the one that starts from the initial configuration $?^E$ (coding the set of all configurations of the PCA). The algorithm stops when at time 0, this trajectory hits the set $\mathcal{A}^E$.

Infinite set of cells. Once again, we consider only one trajectory of the EPCA: at each step , we set $c = ?^{V_{-t}(K)}$ and compute $c = \tilde{\phi}_j(c, (r_i^j)_{i \in V_j(K)}) \in \mathcal{B}^{V_{j+1}(K)}$ for $j = -t$ to -1. This is done until we get $c \in \mathcal{A}^K$.

Algorithm 2: Perfect sampling using the EPCA for a finite set of cells

```
Data: Update function $\tilde{\phi}$. Family
      $(r_k^{-n})_{(k,n) \in E \times \mathbb{N}}$ of i.i.d. r.v.
      uniform on $[0,1]$.
begin
    $t = 1$ ;
    repeat
        $c = ?^E$ ;
        for $j = -t$ to $-1$ do
            $c = \tilde{\phi}(c, (r_i^j)_{i \in E})$
        $t = t + 1$
    until $c \in \mathcal{A}^E$ ;
    return $c$
end
```

▶ **Proposition 4.5.** *The algorithms above (finite and infinite cases) stop almost surely if and only if the EPCA is ergodic. In that case, the output of the algorithm is distributed according to the unique invariant measure of the PCA.*

Proof. The argument is the same in the finite and infinite cases. We give it for the finite case. Assume first that Algorithm 2 stops almost surely. By construction, it implies that for all μ_0, the measure μ_0 env$(P)^n$ is asymptotically supported by $\mathcal{A}^E$. Therefore, we can strengthen the result in Proposition 4.4: the invariant measures of env(P) coincide with the invariant measures of P. In that case, env(P) is ergodic iff P is ergodic. Now recall that the update functions of P and env(P) satisfy (4). Thus, Algorithm 1 also stops almost surely. Furthermore, if we use the same samples $(r_k^{-n})_{(k,n) \in E \times \mathbb{N}}$, Algorithms 1 and 2 will have the same output. According to Proposition 4.1, this output is distributed according to the unique invariant measure of P. In particular, P is ergodic. So env(P) is ergodic.

Assume now that the EPCA is ergodic. The unique invariant measure π of env(P) has to be supported by $\mathcal{A}^E$. Also, by ergodicity, we have $\delta_{?^E}$ env$(P)^n \xrightarrow{w} \pi$. This means precisely that Algorithm 2 stops a.s. ◀

4.3.2 Criteria of ergodicity for the EPCA

▶ **Proposition 4.6.** *Let the set of cells be finite. The EPCA env(P) is ergodic if and only if we have env$(f)(?^V)(?) < 1$. This condition can also be written as:*

$$\min_{x \in \mathcal{A}^V} f(x)(0) + \min_{x \in \mathcal{A}^V} f(x)(1) > 0\,. \tag{5}$$

In particular, on a finite set of cells, if the PCA has positive rates (i.e. $\forall u \in \mathcal{A}^V, \forall a \in \mathcal{A},\ f(u)(a) > 0$), then Algorithm 2 stops a.s.

For an infinite set of cells the situation is more complex. Condition (5) is not sufficient to ensure the ergodicity of the EPCA. A counter-example is given in Section 4.3.3. First, we propose a rough sufficient condition of ergodicity

▶ **Proposition 4.7.** *Let $\alpha^* \in (0,1)$ be the critical probability of the percolation PCA with neighborhood V, see Examples 2.5 and 2.8. The EPCA env(P) is ergodic if*

$$env(f)(?^V)(?) \ < \ \alpha^* \tag{6}$$

and non-ergodic if

$$\min_{x \in \mathcal{B}^V - \mathcal{A}^V} \mathit{env}(f)(x)(?) \ > \ \alpha^* . \tag{7}$$

4.3.3 Counter-examples

Recall Proposition 4.4: [EPCA ergodic] $\implies$ [PCA ergodic]. We now show that the converse is not true.

Let us consider the PCA Majority defined at the beginning of Section 5. For n odd, the PCA is ergodic on the set of cells $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$, by Proposition 5.1. However the associated EPCA satisfies $\mathrm{env}(f)(???) = \delta_?$. According to Proposition 4.6, the EPCA is not ergodic.

Consider the PCA of Example 2.4. This PCA has positive rates, in particular, it satisfies (5). So the EPCA is ergodic on a finite set of cells. Now let the set of cells be $\mathbb{Z}$.

The PCA is ergodic for $\varepsilon \in (0,1)$, see Example 2.7. Consider now the associated EPCA $\mathrm{env}(P)$. Assume for instance that $\varepsilon \in (0, 1/2)$. We have

$$\mathrm{env}(f)(u) = \begin{cases} f(u) & \text{if } u \in \{\mathbf{0},\mathbf{1}\}^V \\ \varepsilon\delta_{\mathbf{0}} + \varepsilon\delta_{\mathbf{1}} + (1-2\varepsilon)\delta_? & \text{otherwise} . \end{cases}$$

By applying Proposition 4.7, $\mathrm{env}(P)$ is non-ergodic if $1 - 2\varepsilon > \alpha^*$.

5 The majority PCA: a case study

The *Majority* PCA is one of the simplest examples of PCA whose behaviour is not well understood. Therefore, it provides a good case study for our sampling algorithms.

Given $0 < \alpha < 1$, the PCA *Majority*(α), or simply *Majority*, is the PCA on the alphabet $\mathcal{A} = \{0, 1\}$, with set of cells $E = \mathbb{Z}$ (or $\mathbb{Z}_n = \mathbb{Z}/n\mathbb{Z}$), neighborhood $V = \{-1, 0, 1\}$, and transition function

$$f(x, y, z) = \alpha\, \delta_{\mathrm{maj}(x,y,z)} + (1 - \alpha)\, \delta_{1-y} ,$$

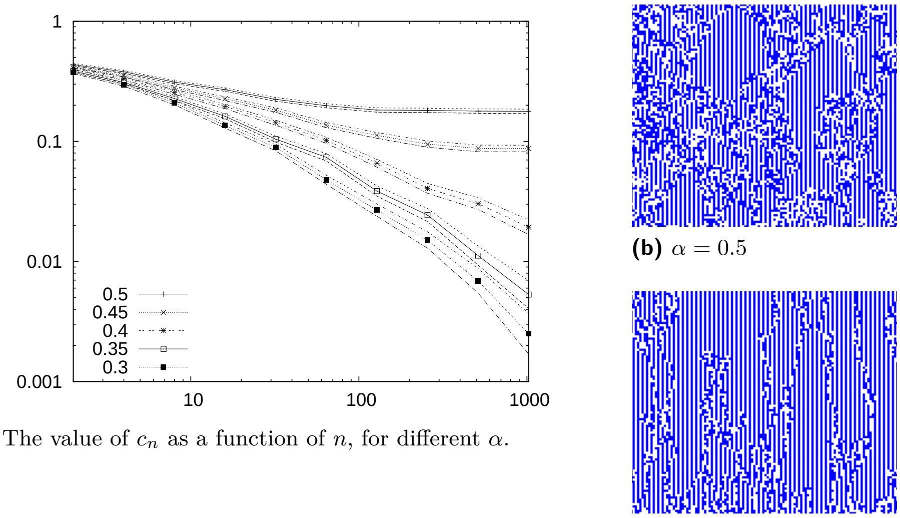
where $\mathrm{maj} : \mathcal{A}^3 \to \mathcal{A}$ is the *majority function*: the value of $\mathrm{maj}(x, y, z)$ is 0, resp. 1, if there are two or three 0's, resp 1's, in the sequence x, y, z. This PCA thus consists in choosing independently for each cell to apply rule 232 (with probability α) or to flip the value.

▶ **Proposition 5.1.** *Consider the Markov chain on the state space $\{0,1\}^{\mathbb{Z}_n}$ which is induced by the Majority PCA on set of cells $\mathbb{Z}_n$. The Markov chain has a unique invariant measure ν. If n is even then $\nu = (\delta_{(01)^{n/2}} + \delta_{(10)^{n/2}})/2$; if n is odd then ν is supported by $\{0,1\}^{\mathbb{Z}_n}$.*

Let us consider now the PCA Majority on $\mathbb{Z}$. Let $x = (01)^{\mathbb{Z}} \in \{0,1\}^{\mathbb{Z}}$ be the configuration defined by: $\forall n \in \mathbb{Z}$, $x_{2n} = 0$, $x_{2n+1} = 1$. The configuration $(10)^{\mathbb{Z}}$ is defined similarly. The probability measure $\mu = (\delta_{(01)^{\mathbb{Z}}} + \delta_{(10)^{\mathbb{Z}}})/2$ is clearly an invariant measure for the PCA Majority. It can be viewed as the "limit" over n of the invariant measures of the PCA on $\mathbb{Z}_{2n}$. What about the "limits" of the invariant measures of the PCA on $\mathbb{Z}_{2n+1}$? Do they define other invariant measures for the PCA on $\mathbb{Z}$?

▶ **Conjecture 5.2.** *There exists $\alpha_c \in (0,1)$ such that* Majority(α) *has a unique invariant measure for $\alpha < \alpha_c$, and several invariant measures for $\alpha > \alpha_c$.*

We propose a partial result relying on ideas of Regnault [13].



(a) The value of c_n as a function of n, for different α.

(b) $\alpha = 0.5$

(c) $\alpha = 0.3$

Figure 2 Experimental study of Majority(α) (the configurations at odd times only are represented on the space-time diagrams).

▶ **Proposition 5.3.** *Let p_c be the percolation threshold of directed bond-percolation in $\mathbb{N}^2$. If $\alpha \geq \sqrt[3]{1-(1-p_c)^4}$, then Majority($\alpha$) has several invariant measures. It is in particular the case if $\alpha \geq 0.996$.*

We also tried to come up with some numerical evidence. To study the PCA Majority experimentally, a first idea would be to consider the same PCA on the set of cells $\mathbb{Z}_n$, n odd, but this does not work well. First, computing exactly the invariant measure is impossible except for small n. Second the efficient perfect sampling is not available since the EPCA is not ergodic.

Instead, we used approximations of the PCA by a (non-homogeneous) PCA on the set of cells $D_n = \{-n, \ldots, n\}$, with random boundary conditions : at each step, the contents of cells $-n$ and n are updated using values of the cells $-(n+1)$ and $n+1$ chosen uniformly at random in $\{0, 1\}$. Again, computing exactly the invariant measure is impossible except for very small windows. But now, the EPCA is ergodic, and the perfect sampling algorithms become effective.

Let μ_n be the unique invariant measure for the set of cells D_n. Define

$$c_n = \mu_n\{x \in X \mid x_0 = x_1 = 0\} + \mu_n\{x \in X \mid x_0 = x_1 = 1\} .$$

One can prove that if $\limsup_n c_n > 0$, then there exists a non-trivial invariant measure for the PCA Majority on $\mathbb{Z}$ (this relies on the compactness of $\mathcal{M}(X)$).

The experimental results appear in Figure 2, with a logarithmic scale. We ran the sampling algorithms 10000 times, up to a window size of $n = 1024$. We show on the figure the confidence intervals calculated with Wilson score test at 95%.

It is reasonable to believe that the top two curves do not converge to 0 while the bottom three converge to 0. This is consistent with the visual impression of space-time diagrams. It reinforces Conjecture 5.2 with a possible phase transition between 0.4 and 0.45.

Acknowledgements This work was partially supported by the ANR project MAGNUM (ANR-2010-BLAN-0204). We used the applet FiatLux developed by N. Fatès (LORIA, INRIA Lorraine) and available on his website to draw the space-time diagrams represented in this article.

References

1 J. van den Berg and J. E. Steif. On the existence and nonexistence of finitary codings for a class of random fields. *Ann. Probab.*, 27(3):1501–1522, 1999.

2 A. Bušić, B. Gaujal, and J.-M. Vincent. Perfect simulation and non-monotone markovian systems. In *Proceedings 3rd Int. Conf. Valuetools*. ICST, 2008.

3 P. Chassaing and J. Mairesse. A non-ergodic probabilistic cellular automaton with a unique invariant measure. arXiv:1009.0143, 2010.

4 K. Culik II, J. Pachl, and S. Yu. On the limit sets of cellular automata. *SIAM J. Comput.*, 18(4):831–842, 1989.

5 N. Fatès, D. Regnault, N. Schabanel, and E. Thierry. Asynchronous behavior of double-quiescent elementary cellular automata. In *LATIN 2006*, volume 3887 of *LNCS*, pages 455–466. Springer, 2006.

6 P. Gács. Reliable cellular automata with self-organization. *J. Statist. Phys.*, 103(1-2):45–267, 2001.

7 G. Grimmett. Percolation. Second edition. Springer-Verlag, Berlin, 1999.

8 P. Guillon and G. Richard. Nilpotency and limit sets of cellular automata. In *MFCS*, volume 5162 of *LNCS*, pages 375–386. Springer, 2008.

9 M. Huber. Perfect sampling using bounding chains. *Ann. Appl. Probab.*, 14(2):734–753, 2004.

10 J. Kari. The nilpotency problem of one-dimensional cellular automata. *SIAM J. Comput.*, 21(3):571–586, 1992.

11 Y. Le Borgne and J.-F. Marckert. Directed animals and gas models revisited. *Electron. J. Combin.*, 14(1):#R71, 2007.

12 J. Propp and D. Wilson. Exact sampling with coupled Markov chains. *Random Structures and Algorithms*, 9:223–252, 1996.

13 D. Regnault. Directed percolation arising in stochastic cellular automata analysis. In *MFCS 2008*, volume 5162 of *LNCS*, pages 563–574. Springer, Berlin, 2008.

14 N. Schabanel. Habilitation à Diriger des Recherches : Systèmes complexes & algorithmes. *Thesis.* LIAFA, CNRS and Université Paris Diderot - Paris 7, 2009.

15 A. Toom. Algorithmical unsolvability of the ergodicity problem for binary cellular automata. *Markov Process. Related Fields*, 6(4):569–577, 2000.

16 A. Toom. Contours, convex sets, and cellular automata. *IMPA Mathematical Publications.* IMPA, Rio de Janeiro, 2001.

17 A. Toom, N. Vasilyev, O. Stavskaya, L. Mityushin, G. Kurdyumov, and S. Pirogov. Discrete local Markov systems. In R. Dobrushin, V. Kryukov, and A. Toom, editors, *Stochastic cellular systems: ergodicity, memory, morphogenesis.* Manchester University Press, 1990.

Analysis of Agglomerative Clustering*

Marcel R. Ackermann[1], Johannes Blömer[1], Daniel Kuntze[1], and Christian Sohler[2]

1 **Department of Computer Science**
University of Paderborn
{mra,bloemer,kuntze}@upb.de

2 **Department of Computer Science**
TU Dortmund
christian.sohler@tu-dortmund.de

Abstract

The diameter k-clustering problem is the problem of partitioning a finite subset of $\mathbb{R}^d$ into k subsets called clusters such that the maximum diameter of the clusters is minimized. One early clustering algorithm that computes a hierarchy of approximate solutions to this problem for all values of k is the agglomerative clustering algorithm with the complete linkage strategy. For decades this algorithm has been widely used by practitioners. However, it is not well studied theoretically. In this paper we analyze the agglomerative complete linkage clustering algorithm. Assuming that the dimension d is a constant, we show that for any k the solution computed by this algorithm is an $O(\log k)$-approximation to the diameter k-clustering problem. Moreover, our analysis does not only hold for the Euclidean distance but for any metric that is based on a norm.

1998 ACM Subject Classification F.2.2 [Analysis of Algorithms and Problem Complexity]: Nonnumerical Algorithms and Problems—Geometrical problems and computations; H.3.3 [Information Storage and Retrieval]: Information Search and Retrieval—Clustering; I.5.3 [Pattern Recognition]: Clustering—Algorithms, Similarity measures

Keywords and phrases agglomerative clustering, hierarchical clustering, complete linkage, approximation guarantees

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.308

1 Introduction

Clustering is the process of partitioning a set of objects into subsets (called clusters) such that each subset contains similar objects and objects in different subsets are dissimilar. It has many applications including data compression [13], analysis of gene expression data [6], anomaly detection [10], and structuring results of search engines [3]. For every application a proper objective function is used to measure the quality of a clustering. One particular objective function is the largest diameter of the clusters. If the desired number of clusters k is given we call the problem of minimizing this objective function the *diameter k-clustering problem.*

One of the earliest and most widely used clustering strategies is agglomerative clustering. The history of agglomerative clustering goes back at least to the 1950s (see for example

* For all four authors this research was supported by the German Research Foundation (DFG), grants BL 314/6-2 and SO 514/4-2.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 308–319
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

[8, 11]). Later, biological taxonomy became one of the driving forces of cluster analysis. In [14] the authors, who where the first biologists using computers to classify organisms, discuss several agglomerative clustering methods.

Agglomerative clustering is a bottom-up clustering process. At the beginning, every input object forms its own cluster. In each subsequent step, the two 'closest' clusters will be merged until only one cluster remains. This clustering process creates a hierarchy of clusters, such that for any two different clusters A and B from possibly different levels of the hierarchy we either have $A \cap B = \emptyset$, $A \subset B$, or $B \subset A$. Such a hierarchy is useful in many applications, for example, when one is interested in hereditary properties of the clusters (as in some bioinformatics applications) or if the exact number of clusters is a priori unknown.

In order to define the agglomerative strategy properly, we have to specify a distance measure between clusters. Given a distance function between data objects, the following distance measures between clusters are frequently used. In the *single linkage strategy*, the distance between two clusters is defined as the distance between their closest pair of data objects. It is not hard to see that this strategy is equivalent to computing a minimum spanning tree of the graph induced by the distance function using Kruskal's algorithm. In case of the *complete linkage strategy*, the distance between two clusters is defined as the distance between their furthest pair of data objects. In the *average linkage strategy* the distance is defined as the average distance between data objects from the two clusters.

1.1 Related Work

In this paper we study the agglomerative clustering algorithm using the complete linkage strategy to find a hierarchical clustering of n points from $\mathbb{R}^d$. The running time is obviously polynomial in the description length of the input. Therefore, our only goal in this paper is to give an approximation guarantee for the diameter k-clustering problem. The approximation guarantee is given by a factor α such that the cost of the k-clustering computed by the algorithm is at most α times the cost of an optimal k-clustering. Although the agglomerative complete linkage clustering algorithm is widely used, only few theoretical results considering the quality of the clustering computed by this algorithm are known. It is known that there exists a certain metric distance function such that this algorithm computes a k-clustering with an approximation factor of $\Omega(\log k)$ [5]. However, prior to the analysis we present in this paper, no non-trivial upper bound for the approximation guarantee of the classical complete linkage agglomerative clustering algorithm was known, and deriving such a bound has been discussed as one of the open problems in [5].

The diameter k-clustering problem is closely related to the *k-center problem*. In this problem, we are searching for k centers and the objective is to minimize the maximum distance of any input point to the nearest center. When the centers are restricted to come from the set of the input points, the problem is called the *discrete k-center problem*. It is known that for metric distance functions the costs of optimal solutions to all three problems are within a factor of 2 from each other.

For the Euclidean case we know that the diameter k-clustering problem and the k-center problem are $\mathcal{NP}$-hard. In fact, it is already $\mathcal{NP}$-hard to approximate both problems with an approximation factor below 1.96 and 1.82 respectively [7].

For fixed k, i.e. when we are not interested in a hierarchy of clusterings, there exist provably good approxiamtion algorithms. For the discrete k-center problem, a simple 2-approximation algorithm is known for metric spaces [9], which immediately yields a 4-approximation algorithm for the diameter k-clustering problem. For the k-center problem, a variety of results is known. For example, for the Euclidean metric in [2] a $(1+\epsilon)$-approximation algorithm with running time $2^{O(k \log k/\epsilon^2)}dn$ is shown. This implies a $(2+\epsilon)$-approximation algorithm with the same running time for the diameter k-clustering problem.

Also, for metric spaces a hierarchical clustering strategy with an approximation guarantee of 8 for the discrete k-center problem is known [5]. This implies an algorithm with an approximation guarantee of 16 for the diameter k-clustering problem.

This paper as well as all of the above mentioned work is about static clustering, i.e. in the problem definition we are given the whole set of input points at once. An alternative model of the input data is to consider sequences of points that are given one after another. In [4] the authors discuss clustering in a so-called *incremental clustering* model. They give an algorithm with constant approximation factor that maintains a hierarchical clustering while new points are added to the input set. Furthermore, they show a lower bound of $\Omega(\log k)$ for the agglomerative complete linkage algorithm and the diameter k-clustering problem. However, since their model differs from ours, this result has no bearing on our lower bounds.

1.2 Our contribution

In this paper, we study the agglomerative complete linkage clustering algorithm for input sets $X \subset \mathbb{R}^d$, where d is constant. To measure the distance between data points, we use a metric that is based on a norm, e.g., the Euclidean metric. We prove that in this case the agglomerative clustering algorithm is an $O(\log k)$-approximation algorithm. Here, the O-notation hides a constant that is doubly exponential in d. This approximation guarantee holds for every level of the hierarchy computed by the algorithm. That is, we compare each computed k-clustering with an optimal solution for the particular value of k. These optimal k-clusterings do not necessarily form a hierarchy. In fact, there are simple examples where optimal solutions have no hierarchical structure.

Our analysis also yields that if we allow $2k$ instead of k clusters and compare the cost of the computed $2k$-clustering to an optimal solution with k clusters, the approximation factor is independent of k and depends only on d. Moreover, the techniques of our analysis can be applied to prove stronger results for the k-center problem and the discrete k-center problem. For the k-center problem we derive an approximation guarantee that is logarithmic in k and only single exponential in d. For the discrete k-center problem we derive an approximation guarantee that is logarithmic in k and the dependence on d is only linear and additive.

Furthermore, we give almost matching upper and lower bounds for the one-dimensional case. These bounds are independent of k. For $d \geq 2$ and the metric based on the ℓ_∞-norm we provide a lower bound that exceeds the upper bound for $d = 1$. For $d \geq 3$ we give a lower bound for the Euclidean case which is above the lower bound for $d = 1$. Finally, we construct instances providing lower bounds for any metric based on an ℓ_p-norm with $1 \leq p \leq \infty$. However, for these instances the lower bounds and the dimension d depend on k.

2 Preliminaries and problem definition

Throughout this paper, we consider input sets that are finite subsets of $\mathbb{R}^d$. Our results hold for arbitrary metrics that are based on a norm, i.e., the distance $\|x-y\|$ between two points $x, y \in \mathbb{R}^d$ is measured using an arbitrary norm $\|\cdot\|$. Readers who are not familiar with arbitrary metrics or are only interested in the Euclidean case, may assume that $\|\cdot\|_2$ is used, i.e. $\|x-y\| = \sqrt{\sum_{i=1}^d (x_i - y_i)^2}$. For $r \in \mathbb{R}$ and $y \in \mathbb{R}^d$ we denote the closed

d-dimensional ball of radius r centered at y by $\mathrm{B}_r^d(y) := \{x \mid \|x - y\| \leq r\}$.

Given $k \in \mathbb{N}$ and a finite set $X \subset \mathbb{R}^d$ with $k \leq |X|$ we say that $\mathcal{C}_k = \{C_1, \ldots, C_k\}$ is a k-clustering of X if the sets $C_1, \ldots, C_k$ (called clusters) form a partition of X into k non-empty subsets. We call a collection of k-clusterings of the same finite set X but for different values of k hierarchical, if it fulfills the following two properties. First, for any $1 \leq k \leq |X|$ the collection contains at most one k-clustering. Second, for any two of its clusterings $\mathcal{C}_i, \mathcal{C}_j$ with $|\mathcal{C}_i| = i < j = |\mathcal{C}_j|$ every cluster in $\mathcal{C}_i$ is the union of one or more clusters from $\mathcal{C}_j$. A hierarchical collection of clusterings is called a hierarchical clustering.

For a finite and non-empty set $C \subset \mathbb{R}^d$ we define the diameter of C to be $\mathrm{diam}(C) := \max_{x,y \in C} \|x - y\|$. Finally, we define the cost of a k-clustering $\mathcal{C}_k$ as its largest diameter, i.e. $\mathrm{cost}(\mathcal{C}_k) := \max_{C \in \mathcal{C}_k} \mathrm{diam}(C)$.

▶ **Problem 1** (diameter k-clustering)**.** Given $k \in \mathbb{N}$ and a finite set $X \subset \mathbb{R}^d$ with $|X| \geq k$ find a k-clustering $\mathcal{C}_k$ of X with minimal cost.

For our analysis of agglomerative clustering we repeatedly use the volume argument stated in Lemma 3. This argument provides an upper bound on the minimum distance between two points from a finite set of points lying inside the union of finitely many balls. For the application of this argument the following definition is crucial.

▶ **Definition 2.** Let $k \in \mathbb{N}$ and $r \in \mathbb{R}$. A set $X \subset \mathbb{R}^d$ is called (k, r)-*coverable* if there exist $y_1, \ldots, y_k \in \mathbb{R}^d$ with $X \subseteq \bigcup_{i=1}^k \mathrm{B}_r^d(y_i)$.

▶ **Lemma 3.** *Let $k \in \mathbb{N}$, $r \in \mathbb{R}$ and $P \subset \mathbb{R}^d$ be finite and (k, r)-coverable with $|P| > k$. Then there exist distinct $p, q \in P$ such that $\|p - q\| \leq 4r \sqrt[d]{\frac{k}{|P|}}$.*

The proof of Lemma 3 can be found in the full version of this paper [1].

3 Analysis

In this section we analyze the agglomerative algorithm for Problem 1 stated as Algorithm 1. Given a finite set $X \subset \mathbb{R}^d$ of input points, the algorithm computes hierarchical k-clusterings for all values of k between 1 and $|X|$. As mentioned before, the algorithm takes a bottom-up approach. It starts with the $|X|$-clustering that contains one cluster for each input point and then successively merges two of the remaining clusters that minimize the diameter of the resulting cluster.

▶ Observation 4. The greedy strategy guarantees that the following holds for all computed clusterings. First, the cost of the clustering is equal to the diameter of the cluster created last. Second, the diameter of the union of any two clusters is always an upper bound for the cost of the clustering to be computed next.

Note that our results hold for any particular tie-breaking strategy. However, to keep the analysis simple, we assume that there are no ties. Thus, for any input set X the clusterings computed by Algorithm 1 are uniquely determined.

Our main result is the following theorem.

▶ **Theorem 5.** *Let $X \subset \mathbb{R}^d$ be a finite set of points. Then for all $k \in \mathbb{N}$ with $k \leq |X|$ the partition $\mathcal{C}_k$ of X into k clusters as computed by Algorithm 1 satisfies*

$$\mathrm{cost}(\mathcal{C}_k) = O(\log k) \cdot \mathrm{opt}_k,$$

where opt_k denotes the cost of an optimal solution to Problem 1, and the constant hidden in the O-notation is doubly exponential in the dimension d.

```
AgglomerativeCompleteLinkage($X$):
  $X$    finite set of input points from $\mathbb{R}^d$
$\mathcal{C}_{|X|} := \{\{x\} \mid x \in X\}$
for $i = |X| - 1, \dots, 1$ do
    find distinct clusters $A, B \in \mathcal{C}_{i+1}$
      minimizing $\mathrm{diam}(A \cup B)$
    $\mathcal{C}_i := (\mathcal{C}_{i+1} \setminus \{A, B\}) \cup \{A \cup B\}$
end for
return $\mathcal{C}_1, \dots, \mathcal{C}_{|X|}$
```

Algorithm 1 The agglomerative complete linkage clustering algorithm.

We prove Theorem 5 in two steps. First, Proposition 6 in Section 3.1 provides an upper bound to the cost of the intermediate $2k$-clustering. This upper bound is independent of k and $|X|$ and may be of independent interest. Second, in the remainder of Section 3, we analyze the k merge steps of Algorithm 1 down to the computation of the k-clustering.

In the following, let $X \subset \mathbb{R}^d$ be the finite set of input points for Algorithm 1 and $k \in \mathbb{N}$ be a fixed number of clusters with $k \leq |X|$. Furthermore, to simplify notation let $r := \mathrm{opt}_k$, where opt_k is the maximum diameter of an optimal solution to Problem 1. Since any cluster C is contained in a ball of radius $\mathrm{diam}(C)$, the set X is (k, r)-coverable, a fact that will be used frequently in our analysis. By $\mathcal{C}_1, \dots, \mathcal{C}_{|X|}$ we denote the clusterings computed by Algorithm 1 on input X.

3.1 Analysis of the $2k$-clustering

▶ **Proposition 6.** *Let $X \subset \mathbb{R}^d$ be finite. Then for all $k \in \mathbb{N}$ with $2k \leq |X|$ the partition $\mathcal{C}_{2k}$ of X into $2k$ clusters as computed by Algorithm 1 satisfies*

$$\mathrm{cost}(\mathcal{C}_{2k}) < 2^{3\sigma}\,(28d + 6) \cdot \mathrm{opt}_k,$$

where $\sigma = (42d)^d$ and opt_k denotes the cost of an optimal solution to Problem 1.

To prove Proposition 6 we divide the merge steps of Algorithm 1 into two stages. The first stage consists of the merge steps down to a $2^{2^{O(d \log d)}} k$-clustering. The analysis of the first stage is based on the following notion of similarity. Two clusters are called similar if one cluster can be translated such that every point of the translated cluster is near a point of the second cluster. Then, by merging similar clusters, the diameter essentially increases by the length of the translation vector. During the whole first stage we guarantee that there is a sufficiently large number of similar clusters left. The cost of the intermediate $2^{2^{O(d \log d)}} k$-clustering can be upper bounded by $O(d) \cdot \mathrm{opt}_k$.

The second stage consists of the merge steps reducing the number of remaining clusters from $2^{2^{O(d \log d)}} k$ to only $2k$. In this stage we are no longer able to guarantee that a sufficiently large number of similar clusters exists. Therefore, we analyze the merge steps of the second stage using a weaker argument. The underlying reasoning of what we do for the second stage is the following. If there are more than $2k$ clusters left, we are able to find sufficiently many pairs of clusters that intersect with the same cluster of an optimal k-clustering. As long as one of these pairs is left, the cost of merging this pair gives an upper bound on the cost of the next merge step. Therefore, we can bound the diameter of the created cluster by the sum of the diameters of the two clusters plus the diameter of the optimal cluster. We find that the cost of the intermediate $2k$-clustering is upper bounded by $2^{2^{O(d \log d)}} \cdot \mathrm{opt}_k$. Let us remark that we do not obtain our main result if we already use this argument for the first stage.

3.2 Stage one

In our analysis the first stage is subdivided into phases, such that in each phase the number of remaining clusters is reduced by one fourth. The following lemma will be used to bound the increase of the cost during a single phase.

▶ **Lemma 7.** *Let $\lambda \in \mathbb{R}$ with $0 < \lambda < 1$ and $\rho := \left\lceil \left(\frac{3}{\lambda}\right)^d \right\rceil$. Furthermore let $m \in \mathbb{N}$ with $2^{\rho+1}k < m \leq |X|$. Then*

$$\operatorname{cost}(\mathcal{C}_{\lfloor \frac{3m}{4} \rfloor}) < (1 + 2\lambda) \cdot \operatorname{cost}(\mathcal{C}_m) + 4r \sqrt[d]{\frac{2^{\rho+1}k}{m}}. \tag{1}$$

Proof. Let $t := \left\lfloor \frac{3m}{4} \right\rfloor$ and $\mathcal{S} := \mathcal{C}_m \cap \mathcal{C}_{t+1}$ be the set of clusters from $\mathcal{C}_m$ that still exist $\left\lceil \frac{m}{4} \right\rceil - 1$ merge steps after the computation of $\mathcal{C}_m$. In each iteration of its loop, the algorithm can merge at most two clusters from $\mathcal{C}_m$. Thus $|\mathcal{S}| > \frac{m}{2}$.

From every cluster $C \in \mathcal{S}$ we fix an arbitrary point and denote it by p_C. Let $R := \operatorname{cost}(\mathcal{C}_m)$. Then the distance from p_C to any $q \in C$ is at most R and we get $C - p_C \subset \mathrm{B}^d_R(0)$.

A ball of radius R can be covered by ρ balls of radius λR (see [12]). Hence, there exist $y_1, \ldots, y_\rho \in \mathbb{R}^d$ with $\mathrm{B}^d_R(0) \subseteq \bigcup_{i=1}^{\rho} \mathrm{B}^d_{\lambda R}(y_i)$. For $C \in \mathcal{S}$ we call the set $\operatorname{Conf}(C) := \{y_i \mid 1 \leq i \leq \rho \text{ and } \mathrm{B}^d_{\lambda R}(y_i) \cap (C - p_C) \neq \varnothing\}$ the configuration of C. That is, we indentify each cluster $C \in \mathcal{S}$ with the subset of the balls $\mathrm{B}^d_{\lambda R}(y_1), \ldots, \mathrm{B}^d_{\lambda R}(y_\rho)$ that intersect with $C - p_C$. Note that no cluster from $C \in \mathcal{S}$ has an empty configuration. The number of possible configurations can be upper bounded by 2^ρ. With $|\mathcal{S}| > \frac{m}{2}$ it follows that there exist $j > \frac{m}{2^{\rho+1}}$ distinct clusters $C_1, \ldots, C_j \in \mathcal{S}$ with the same configuration. Using $m > 2^{\rho+1}k$ we deduce $j > k$.

Let $P := \{p_{C_1}, \ldots, p_{C_j}\}$. Since X is (k, r)-coverable, so is $P \subset X$. Therefore, by Lemma 3, there exist distinct $a, b \in \{1, \ldots, j\}$ such that $\|p_{C_a} - p_{C_b}\| \leq 4r \sqrt[d]{\frac{2^{\rho+1}k}{m}}$.

Next we want to bound the diameter of the union of the corresponding clusters C_a and C_b. The distance between any two points $u, v \in C_a$ or $u, v \in C_b$ is at most the cost of $\mathcal{C}_m$. Now let $u \in C_a$ and $v \in C_b$. Using the triangle inequality, for any $w \in \mathbb{R}^d$ we obtain $\|u - v\| \leq \|p_{C_a} - p_{C_b}\| + \|u + p_{C_b} - p_{C_a} - w\| + \|w - v\|$.

For $\|p_{C_a} - p_{C_b}\|$ we just derived an upper bound. To bound $\|u + p_{C_b} - p_{C_a} - w\|$, we let $y \in \operatorname{Conf}(C_a) = \operatorname{Conf}(C_b)$ such that $u - p_{C_a} \in \mathrm{B}^d_{\lambda R}(y)$. Furthermore, we fix $w \in C_b$ with $w - p_{C_b} \in \mathrm{B}^d_{\lambda R}(y)$. Hence, $\|u + p_{C_b} - p_{C_a} - w\| = \|u - p_{C_a} - (w - p_{C_b})\|$ can be upper bounded by $2\lambda R = 2\lambda \cdot \operatorname{cost}(\mathcal{C}_m)$. For $w \in C_b$ the distance $\|w - v\|$ is bounded by $\operatorname{diam}(C_b) \leq \operatorname{cost}(\mathcal{C}_m)$. We conclude that merging clusters C_a and C_b results in a cluster whose diameter can be upper bounded by

$$\operatorname{diam}(C_a \cup C_b) < (1 + 2\lambda) \cdot \operatorname{cost}(\mathcal{C}_m) + 4r \sqrt[d]{\frac{2^{\rho+1}k}{m}}.$$

Using Observation 4 and the fact that C_a and C_b are part of the clustering $\mathcal{C}_{t+1}$, we can upper bound the cost of $\mathcal{C}_t$ by $\operatorname{cost}(\mathcal{C}_t) \leq \operatorname{diam}(C_a \cup C_b)$. ◀

Note that the parameter λ from Lemma 7 establishes a trade-off between the two terms on the right-hand side of Inequality 1. To complete the analysis of the first stage, we have to carefully choose λ. In the proof of the following lemma we use $\lambda = {}^{\ln \frac{4}{3}}/_{4d}$ and apply Lemma 7 for $\left\lceil \log_{\frac{4}{3}} \frac{|X|}{2^{\sigma+1}k} \right\rceil$ consecutive phases, where $\sigma = (42d)^d$. Then, we are able to upper bound the total increase of the cost by a term that is linear in d and r and independent of $|X|$ and k. The number of remaining clusters is independent of the number of input points $|X|$ and depends only on the dimension d and the desired number of clusters k.

▶ **Lemma 8.** *Let $2^{\sigma+1}k < |X|$ for $\sigma = (42d)^d$. Then on input X Algorithm 1 computes a clustering $\mathcal{C}_{2^{\sigma+1}k}$ with* $\operatorname{cost}(\mathcal{C}_{2^{\sigma+1}k}) < (28d+4) \cdot r$.

Proof. Let $u := \left\lceil \log_{\frac{3}{4}} \frac{2^{\sigma+1}k}{|X|} \right\rceil$ and define $m_i := \left\lceil \left(\frac{3}{4}\right)^i |X| \right\rceil$ for all $i = 0, \dots, u$. Furthermore let $\lambda = \ln \frac{4}{3}/4d$. This implies $\rho \le \sigma$ for the parameter ρ of Lemma 7. Then $m_u \le 2^{\sigma+1}k$ and $m_i > 2^{\sigma+1}k \ge 2^{\rho+1}k$ for all $i = 0, \dots, u-1$. We apply Lemma 7 with $m = m_i$ for all $i = 0, \dots, u-1$. Since $\left\lfloor \frac{3m_i}{4} \right\rfloor \le m_{i+1}$ and Algorithm 1 uses a greedy strategy we deduce $\operatorname{cost}(\mathcal{C}_{m_{i+1}}) \le \operatorname{cost}(\mathcal{C}_{\lfloor \frac{3m_i}{4} \rfloor})$ for all $i = 0, \dots, u-1$. Using $\operatorname{cost}(\mathcal{C}_{2^{\sigma+1}k}) \le \operatorname{cost}(\mathcal{C}_{m_u})$ and $\operatorname{cost}(\mathcal{C}_{m_0}) = 0$ we get

$$\begin{aligned}
\operatorname{cost}\left(\mathcal{C}_{2^{\sigma+1}k}\right) &< \sum_{i=0}^{u-1} \left((1+2\lambda)^i \cdot 4r \sqrt[d]{\frac{2^{\sigma+1}k}{\left(\frac{3}{4}\right)^{u-1-i} |X|}} \right) \\
&= 4r \sqrt[d]{\frac{2^{\sigma+1}k}{\left(\frac{3}{4}\right)^{u-1} |X|}} \cdot \sum_{i=0}^{u-1} \left((1+2\lambda)^i \cdot \sqrt[d]{\left(\frac{3}{4}\right)^i} \right).
\end{aligned}$$

Using $u - 1 < \log_{\frac{3}{4}} \frac{2^{\sigma+1}k}{|X|}$ we get

$$\operatorname{cost}\left(\mathcal{C}_{2^{\sigma+1}k}\right) < 4r \sum_{i=0}^{u-1} \left(\frac{1+2\lambda}{\sqrt[d]{\frac{4}{3}}} \right)^i. \tag{2}$$

By taking only the first two terms of the series expansion of the exponential function we get $1 + 2\lambda = 1 + \frac{\ln \frac{4}{3}}{2d} < e^{\frac{\ln \frac{4}{3}}{2d}} = \sqrt[2d]{\frac{4}{3}}$. Substituting this bound into Inequality (2) and extending the sum gives

$$\operatorname{cost}\left(\mathcal{C}_{2^{\sigma+1}k}\right) < 4r \sum_{i=0}^{\infty} \left(\frac{1}{\sqrt[2d]{\frac{4}{3}}} \right)^i < 4r \sum_{i=0}^{\infty} \left(\frac{1}{1+2\lambda} \right)^i.$$

Solving the geometric series leads to

$$\operatorname{cost}\left(\mathcal{C}_{2^{\sigma+1}k}\right) < 4r \left(\frac{1}{2\lambda} + 1 \right) < (28d+4) \cdot r.$$

◀

3.3 Stage two

The second stage covers the remaining merge steps until Algorithm 1 computes the clustering $\mathcal{C}_{2k}$. The following lemma is the analogon of Lemma 8. Again, the proof subdivides the merge steps into phases of one fourth of the remaining steps. However, compared to stage one, the analysis of a single phase yields a weaker bound. The proof can be found in the full version of this paper [1].

▶ **Lemma 9.** *Let $n \in \mathbb{N}$ with $n \le 2^{\sigma+1}k$ and $2k < n \le |X|$ for $\sigma = (42d)^d$. Then on input X Algorithm 1 computes a clustering $\mathcal{C}_{2k}$ with*

$$\operatorname{cost}(\mathcal{C}_{2k}) < 2^{3\sigma} \left(\operatorname{cost}(\mathcal{C}_n) + 2r \right).$$

Proposition 6 follows immediately by combining Lemma 8 and Lemma 9.

3.4 Connected instances

For the analysis of the two stages in Section 3.1 we use arguments that are only applicable if there are enough clusters left (at least $2k$ in case of stage two). To analyze the remaining merge steps, we show that it is sufficient to analyze Algorithm 1 on a subset $Y \subseteq X$ satisfying a certain connectivity property. Using this property we are able to apply a combinatorial approach that relies on the number of merge steps left. This introduces the $O(\log k)$ term to the approximation factor of our main result.

We start by defining the connectivity property that will be used to relate clusters to an optimal k-clustering.

▶ **Definition 10.** Let $Z \subseteq \mathbb{R}^d$ and $r \in \mathbb{R}$. Two sets $A, B \subseteq \mathbb{R}^d$ are called (Z, r)-*connected* if there exists a $z \in Z$ with $\mathrm{B}^d_r(z) \cap A \neq \varnothing$ and $\mathrm{B}^d_r(z) \cap B \neq \varnothing$.

Note that for any two (Z, r)-connected clusters A, B we have

$$\mathrm{diam}(A \cup B) \leq \mathrm{diam}(A) + \mathrm{diam}(B) + 2r. \tag{3}$$

Next, we show that for any input set X we can bound the cost of the k-clustering computed by Algorithm 1 by the cost of the ℓ-clustering computed by the algorithm on a connected subset $Y \subseteq X$ for a proper $\ell \leq k$. Recall that by our convention from the beginning of Section 3, the clusterings computed by Algorithm 1 on a particular input set are uniquely determined.

▶ **Lemma 11.** *Let $X \subset \mathbb{R}^d$ be finite and $k \in \mathbb{N}$ with $k \leq |X|$. Then there exists a subset $Y \subseteq X$, a number $\ell \in \mathbb{N}$ with $\ell \leq \min(k, |Y|)$, and a set $Z \subset \mathbb{R}^d$ with $|Z| = \ell$ such that:*

1. *Y is (ℓ, r)-coverable;*
2. *$\mathrm{cost}(\mathcal{C}_k) \leq \mathrm{cost}(\mathcal{P}_\ell)$;*
3. *For all $n \in \mathbb{N}$ with $\ell + 1 \leq n \leq |Y|$, every cluster in $\mathcal{P}_n$ is (Z, r)-connected to another cluster in $\mathcal{P}_n$.*

Here, the collection $\mathcal{P}_1, \ldots, \mathcal{P}_{|Y|}$ denotes the hierarchical clustering computed by Algorithm 1 on input Y.

Proof. To define Y, Z, and ℓ we consider the $(k+1)$-clustering computed by Algorithm 1 on input X. We know that $X = \bigcup_{A \in \mathcal{C}_{k+1}} A$ is (k, r)-coverable. Let $E \subseteq \mathcal{C}_{k+1}$ be a minimal subset such that $\bigcup_{A \in E} A$ is $(|E| - 1, r)$-coverable, i.e., for all sets $F \subseteq \mathcal{C}_{k+1}$ with $|F| < |E|$ the union $\bigcup_{A \in F} A$ is not $(|F| - 1, r)$-coverable. Since a set F of size 1 cannot be $(|F| - 1, r)$-coverable, we get $|E| \geq 2$.

Let $Y := \bigcup_{A \in E} A$ and $\ell := |E| - 1$. Then $\ell \leq k$ and Y is (ℓ, r)-coverable. Thus, we can define $Z \subset \mathbb{R}^d$ with $|Z| = \ell$ and $Y \subset \bigcup_{z \in Z} \mathrm{B}^d_r(z)$. Furthermore, we let $\mathcal{P}_1, \ldots, \mathcal{P}_{|Y|}$ be the hierarchical clustering computed by Algorithm 1 on input Y.

Since Y is the union of the clusters from $E \subseteq \mathcal{C}_{k+1}$, each merge step between the computation of $\mathcal{C}_{|X|}$ and $\mathcal{C}_{k+1}$ merges either two clusters $A, B \subset Y$ or two clusters $A, B \subset X \backslash Y$. The merge steps inside $X \backslash Y$ have no influence on the clusters inside Y. Furthermore, the merge steps inside Y would be the same in the absence of the clusters inside $X \setminus Y$. Therefore, on input Y Algorithm 1 computes the $(\ell + 1)$-clustering $\mathcal{P}_{\ell+1} = E = \mathcal{C}_{k+1} \cap 2^Y$. Thus, $\mathcal{P}_{\ell+1} \subseteq \mathcal{C}_{k+1}$.

To compute $\mathcal{P}_\ell$, Algorithm 1 on input Y merges two clusters from $\mathcal{P}_{\ell+1}$ that minimize the diameter of the resulting cluster. Analogously, Algorithm 1 on input X merges two clusters from $\mathcal{C}_{k+1}$ to compute $\mathcal{C}_k$. Since $\mathcal{P}_{\ell+1} \subseteq \mathcal{C}_{k+1}$, Observation 4 implies $\mathrm{cost}(\mathcal{C}_k) \leq \mathrm{cost}(\mathcal{P}_\ell)$.

It remains to show that for all $n \in \mathbb{N}$ with $\ell + 1 \leq n \leq |Y|$ it holds that every cluster in $\mathcal{P}_n$ is (Z, r)-connected to another cluster in $\mathcal{P}_n$. We first show the property for $n = \ell + 1$.

For $\ell = 1$ this follows from the fact that $\mathrm{B}_r^d(z)$ with $Z = \{z\}$ has to contain both clusters from $\mathcal{P}_2$. For $\ell > 1$ we are otherwise able to remove one cluster from $\mathcal{P}_{\ell+1}$ and get ℓ clusters whose union is $(\ell - 1, r)$-coverable. This contradicts the definition of $E = \mathcal{P}_{\ell+1}$ as a minimal subset with this property.

To prove 3. for general n, let $C_1 \in \mathcal{P}_n$ and $z \in Z$ with $\mathrm{B}_r^d(z) \cap C_1 \neq \varnothing$. There exists a unique cluster $\tilde{C}_1 \in \mathcal{P}_{\ell+1}$ with $C_1 \subseteq \tilde{C}_1$. Then we have $\mathrm{B}_r^d(z) \cap \tilde{C}_1 \neq \varnothing$. However, $\mathrm{B}_r^d(z)$ has to intersect with at least two clusters from $\mathcal{P}_{\ell+1}$. Thus, there exists another cluster $\tilde{C}_2 \in \mathcal{P}_{\ell+1}$ with $\mathrm{B}_r^d(z) \cap \tilde{C}_2 \neq \varnothing$. Since every cluster from $\mathcal{P}_{\ell+1}$ is a union of clusters from $\mathcal{P}_n$, there exists at least one cluster $C_2 \in \mathcal{P}_n$ with $C_2 \subseteq \tilde{C}_2$ and $\mathrm{B}_r^d(z) \cap C_2 \neq \varnothing$. ◀

3.5 Analysis of the remaining merge steps

Let Y, Z, ℓ, and $\mathcal{P}_1, \dots, \mathcal{P}_{|Y|}$ be as given in Lemma 11. Then, Proposition 6 can be used to obtain an upper bound for the cost of $\mathcal{P}_{2\ell}$. In the following, we analyze the merge steps leading from $\mathcal{P}_{2\ell}$ to $\mathcal{P}_{\ell+1}$ and show how to obtain an upper bound for the cost of $\mathcal{P}_{\ell+1}$. As in Section 3.1, we analyze the merge steps in phases. The following lemma is used to bound the increase of the cost during a single phase.

▶ **Lemma 12.** *Let $m, n \in \mathbb{N}$ with $n \leq 2\ell$ and $\ell < m \leq n \leq |Y|$. If there are no two (Z, r)-connected clusters in $\mathcal{P}_m \cap \mathcal{P}_n$, it holds*

$$\mathrm{cost}(\mathcal{P}_{\lfloor \frac{m+\ell}{2} \rfloor}) \leq \mathrm{cost}(\mathcal{P}_m) + 2 \cdot (\mathrm{cost}(\mathcal{P}_n) + 2r)\,.$$

Proof. We show that there exist at least $m - \ell$ disjoint pairs of clusters from $\mathcal{P}_m$ such that the diameter of their union can be upper bounded by $\mathrm{cost}(\mathcal{P}_m) + 2 \cdot (\mathrm{cost}(\mathcal{P}_n) + 2r)$. By Observation 4, this upper bounds the cost of the computed clusterings as long as such a pair of clusters remains. Then the lemma follows from the fact that in each iteration of its loop the algorithm can destroy at most two of these pairs.

To bound the number of such pairs of clusters we start with a structural observation. Let $\mathcal{S} := \mathcal{P}_m \cap \mathcal{P}_n$ be the set of clusters from $\mathcal{P}_n$ that still exist in $\mathcal{P}_m$. By our definiton of Y, Z, and ℓ we find that any cluster $A \in \mathcal{S} \subseteq \mathcal{P}_m$ is (Z, r)-connected to another cluster $B \in \mathcal{P}_m$. If we assume that there are no two (Z, r)-connected clusters in $\mathcal{S}$, this implies $B \in \mathcal{P}_m \setminus \mathcal{S}$. Thus, using $A \in \mathcal{P}_n$, $B \in \mathcal{P}_m$, and Equation (3) the diameter of $A \cup B$ can be bounded by

$$\mathrm{diam}(A \cup B) \leq \mathrm{cost}(\mathcal{P}_m) + \mathrm{cost}(\mathcal{P}_n) + 2r. \tag{4}$$

Moreover, using similar argument, if two clusters $A_1, A_2 \in \mathcal{S} \subseteq \mathcal{P}_n$ are (Z, r)-connected to the same cluster $B \in \mathcal{P}_m \setminus \mathcal{S}$ we can bound the diameter of $A_1 \cup A_2$ by

$$\mathrm{diam}(A_1 \cup A_2) \leq \mathrm{cost}(\mathcal{P}_m) + 2 \cdot (\mathrm{cost}(\mathcal{P}_n) + 2r). \tag{5}$$

Next we show that there exist at least $\left\lceil \frac{|\mathcal{S}|}{2} \right\rceil$ disjoint pairs of clusters from $\mathcal{P}_m$ such that the diameter of their union can be bounded either by Inequality (4) or by Inequality (5). To do so, we first consider the pairs of clusters from $\mathcal{S}$ that are (Z, r)-connected to the same cluster from $\mathcal{P}_m \setminus \mathcal{S}$ until no candidates are left. For these pairs we can bound the diameter of their union by Inequality (5). Then, each cluster from $\mathcal{P}_m \setminus \mathcal{S}$ is (Z, r)-connected to at most one of the remaining clusters from $\mathcal{S}$. Thus, each remaining cluster $A \in \mathcal{S}$ can be paired with a different cluster $B \in \mathcal{P}_m \setminus \mathcal{S}$ such that A and B are (Z, r)-connected. For these pairs we can bound the diameter of their union by Inequality (4). Since for all pairs either one or both of the clusters come from the set $\mathcal{S}$, we can lower bound the number of pairs by $\left\lceil \frac{|\mathcal{S}|}{2} \right\rceil$.

To complete the proof, we show that $m - \ell \leq \left\lceil \frac{|\mathcal{S}|}{2} \right\rceil$. In each iteration of its loop the algorithm can merge at most two clusters from $\mathcal{P}_n$. Therefore, to compute $\mathcal{P}_m$, at least $\left\lceil \frac{n-|\mathcal{S}|}{2} \right\rceil$ merge steps must have been done since the computation of $\mathcal{P}_n$. Hence, $m \leq n - \left\lceil \frac{n-|\mathcal{S}|}{2} \right\rceil \leq \frac{n}{2} + \frac{|\mathcal{S}|}{2}$. Using $n \leq 2\ell$ we get $m - \ell \leq \frac{|\mathcal{S}|}{2}$. ◀

▶ **Lemma 13.** *Let $n \in \mathbb{N}$ with $n \leq 2\ell$ and $\ell < n \leq |Y|$. Then*

$$\operatorname{cost}(\mathcal{P}_{\ell+1}) < 2(\log_2 \ell + 2) \cdot (\operatorname{cost}(\mathcal{P}_n) + 2r)\,.$$

Proof. For $n = \ell + 1$ there is nothing to show. Hence, assume $n > \ell + 1$. Then by definition of Z there exist two (Z, r)-connected clusters in $\mathcal{P}_n$. Now let $\tilde{n} \in \mathbb{N}$ with $\tilde{n} < n$ be maximal such that no two (Z, r)-connected clusters exist in $\mathcal{P}_{\tilde{n}} \cap \mathcal{P}_n$. The number $\tilde{n}$ is well-defined since at least the set $\mathcal{P}_1$ does not contain two clusters at all. It follows that the same holds for all $m \in \mathbb{N}$ with $m \leq \tilde{n}$. We conclude that Lemma 12 is applicable for all $m \leq \tilde{n}$ with $\ell < m$.

By the definition of $\tilde{n}$ there still exist at least two (Z, r)-connected clusters in $\mathcal{P}_{\tilde{n}+1} \cap \mathcal{P}_n$. Then, Observation 4 implies

$$\operatorname{cost}(\mathcal{P}_{\tilde{n}}) \leq 2 \cdot \operatorname{cost}(\mathcal{P}_n) + 2r. \tag{6}$$

If $\tilde{n} \leq \ell + 1$ then Inequality (6) proves the lemma. For $\tilde{n} > \ell + 1$ let $u := \lceil \log_2(\tilde{n} - \ell) \rceil$ and define $m_i := \left\lceil \left(\frac{1}{2}\right)^i (\tilde{n} - \ell) + \ell \right\rceil > \ell$ for all $i = 0, \ldots, u$. We apply Lemma 12 with $m = m_i$ for all $i = 0, \ldots, u-1$. Since $\left\lfloor \frac{m_i + \ell}{2} \right\rfloor \leq m_{i+1}$ and Algorithm 1 uses a greedy strategy we deduce $\operatorname{cost}(\mathcal{C}_{m_{i+1}}) \leq \operatorname{cost}(\mathcal{C}_{\left\lfloor \frac{m_i+\ell}{2} \right\rfloor})$ for all $i = 0, \ldots, u-1$. By summation over all i, we get

$$\operatorname{cost}(\mathcal{P}_{m_u}) < \operatorname{cost}(\mathcal{P}_{\tilde{n}}) + 2u \cdot (\operatorname{cost}(\mathcal{P}_n) + 2r) \overset{(6)}{<} 2(u+1) \cdot (\operatorname{cost}(\mathcal{P}_n) + 2r)\,.$$

Since $\tilde{n} < 2\ell$ we get $u < \log_2 \ell + 1$ and the lemma follows using $m_u = \ell + 1$. ◀

The following lemma finishes the analysis except for the last merge step.

▶ **Lemma 14.** *Let $Y \subset \mathbb{R}^d$ be finite and $\ell \leq |Y|$ such that Y is (ℓ, r)-coverable. Furthermore, let $Z \subset \mathbb{R}^d$ with $|Z| = \ell$ such that for all $n \in \mathbb{N}$ with $\ell + 1 \leq n \leq |Y|$ every cluster in $\mathcal{P}_n$ is (Z, r)-connected to another cluster in $\mathcal{P}_n$, where $\mathcal{P}_1, \ldots, \mathcal{P}_{|Y|}$ denotes the hierarchical clustering computed by Algorithm 1 on input Y. Then $\operatorname{cost}(\mathcal{P}_{\ell+1}) < 2(\log_2 \ell + 2) \cdot \left(2^{3\sigma}(28d + 6) + 2\right) \cdot r$ for $\sigma = (42d)^d$.*

Proof. Let $n := \min(|Y|, 2\ell)$. Then, using Proposition 6, we get $\operatorname{cost}(\mathcal{P}_n) < 2^{3\sigma}(28d + 6) \cdot r$. The lemma follows by using this bound in combination with Lemma 13. ◀

3.6 Proof of Theorem 5

Using Lemma 11 we know that there is a subset $Y \subseteq X$, a number $\ell \leq k$, and a hierarchical clustering $\mathcal{P}_1, \ldots, \mathcal{P}_{|Y|}$ of Y with $\operatorname{cost}(\mathcal{C}_k) \leq \operatorname{cost}(\mathcal{P}_\ell)$. Furthermore, there is a set $Z \subset \mathbb{R}^d$ such that every cluster from $\mathcal{P}_{\ell+1}$ is (Z, r)-connected to another cluster in $\mathcal{P}_{\ell+1}$. Thus, $\mathcal{P}_{\ell+1}$ contains two clusters A, B that intersect with the same ball of radius r. Hence $\operatorname{cost}(\mathcal{C}_k) \leq \operatorname{diam}(A \cup B) \leq 2 \cdot \operatorname{cost}(\mathcal{P}_{\ell+1}) + 2r$. The theorem follows using Lemma 14 and $\ell \leq k$. ◀

4 Further results and open problems

4.1 Lower bounds

For $d = 1$ we are able to show that Algorithm 1 computes an approximation to Problem 1 with an approximation factor between 2.5 and 3. We even know that for any input set $X \subset \mathbb{R}$ the approximation factor of the computed solution is strictly below 3. However, we do not show an approximation factor of $3 - \epsilon$ for some $\epsilon > 0$. The proof of the upper bound is very technical, makes extensive use of the total order of the real numbers, and is too long to be included in this extended abstract.

Furthermore, we know that the dimension d has an impact on the approximation factor of Algorithm 1. This follows from a 2-dimensional input set yielding a lower bound of 3 for the metric based on the ℓ_∞-norm. Note that this exceeds the upper bound from the one-dimensional case. Furthermore, for the Euclidean metric we know a 3-dimensional input set giving a lower bound of 2.56, thus exceeding our lower bound from the one-dimensional case.

Moreover, we can show that there exist input instances such that Algorithm 1 computes an approximation to Problem 1 with an approximation factor of $\Omega(\sqrt[p]{\log k})$ for metrics based on an ℓ_p-norm ($1 \leq p < \infty$) and $\Omega(\log k)$ for the metric based on the ℓ_∞-norm. In case of the ℓ_1- and the ℓ_∞-norm this matches the already known lower bound [5] that has been shown using a rather artificial metric. However, in our instances the dimension d is not fixed but depends on k.

All lower bounds mentioned above are proven in the full version of this paper [1].

4.2 Related clustering problems

As mentioned in the introduction, the cost of optimal solutions to the diameter k-clustering problem, the k-center problem, and the discrete k-center problem are within a factor of 2 from each other. That is, Algorithm 1 computes an $O(\log k)$-approximation for all three problems.

In case of the k-center problem and the discrete k-center problem, our techniques can be applied in a simplified way and yield stronger bounds. Here, we consider the agglomerative algorithm that minimizes the (discrete) k-center cost function in every merge step. In case of the k-center problem we are able to show an upper bound that is logarithmic in k and single exponential in d. More precisely, the dependency on d in the upper bound for the cost of the $2k$-clustering from doubly exponential to only single exponential. Mainly this is achieved because the analysis no longer requires configurations of clusters.

In case of the discrete k-center problem we are able to show an upper bound of $20d + 2\log_2 k + 4$ for the approximation factor. Here, the analysis benefits from the fact that we are able to bound the increase of the cost in each phase of the second stage by a term that is only additive.

The lower bound of $\Omega(\sqrt[p]{\log k})$ for any ℓ_p-norm and $\Omega(\log k)$ for the ℓ_∞-norm can be adopted to the discrete k-center problem (see full version of this paper [1]). In particular, in case of the ℓ_2-norm we obtain an instance in dimension $O(\log^3 k)$ for which we can show a lower bound of $\Omega(\sqrt{\log k})$. Applying the upper bound of $20d + 2\log_2 k + 4$ to this instance we see that Algorithm 1 computes a k-clustering whose cost is $O(\log^3 k)$ times the cost of an optimal solution. This implies that the approximation factor of Algorithm 1 cannot be simultaneously independent of d and $\log k$. More precisely, the approximation factor cannot be sublinear in $\sqrt[6]{d}$ and in $\sqrt{\log k}$.

4.3 Open problems

The main open problems our contribution raises are:

- Can the doubly exponential dependence on d in Theorem 5 be improved?
- Are the different dependencies on d in the approximation factors for the discrete k-center problem, the k-center problem, and the diameter k-clustering problem due to the limitations of our analysis or are they inherent to these problems?
- Can our results be extended to more general distance measures?
- Can the lower bounds for ℓ_p-metrics with $1 < p < \infty$ be improved to $\Omega(\log k)$, matching the lower bound from [5] for all ℓ_p-norms?

References

1 M. R. Ackermann, J. Blömer, D. Kuntze, and C. Sohler. Analysis of Agglomerative Clustering. *CoRR: Computing Research Repository*, 2010. `arXiv:1012.3697 [cs.DS]`.

2 M. Badoiu, S. Har-Peled, and P. Indyk. Approximate Clustering via Core-Sets. In *Proceedings of STOC '02*, pages 250–257, 2002.

3 A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic Clustering of the Web. In *Selected papers from the sixth intern. conf. on WWW*, pages 1157–1166, Essex, UK, 1997. Elsevier Science Publishers Ltd.

4 M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental Clustering and Dynamic Information Retrieval. *SIAM J. Comput.*, 33:1417–1440, June 2004.

5 S. Dasgupta and P. M. Long. Performance Guarantees for Hierarchical Clustering. *JCSS: Journal of Computer and System Sciences*, 70(4):555–569, 2005.

6 M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *PNAS: Proceedings of the National Academy of Sciences*, 95(25):14863–14868, December 1998.

7 T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Proceedings of STOC '88*, pages 434–444, New York, NY, USA, 1988. ACM.

8 K. Florek, J. Lukaszewicz, J. Perkal, H. Steinhaus, and S. Zubrzycki. Sur la liaison et la division des points d'un ensemble fini. *Colloqium Math.*, 2:282–285, 1951.

9 T. F. Gonzalez. Clustering to Minimize the Maximum Intercluster Distance. *Theoretical Computer Science*, 38:293–306, 1985.

10 K. Lee, J. Kim, K. Kwon, Y. Han, and S. Kim. DDoS attack detection method using cluster analysis. *Expert Systems with Applications*, 34(3):1659–1665, 2008.

11 L. L. McQuitty. Elementary Linkage Analysis for Isolating Orthogonal and Oblique Types and Typal Relevancies. *Educational and Psychological Measurement*, 17:207–209, 1957.

12 M. Naszódi. Covering a Set with Homothets of a Convex Body. *Positivity*, 2009.

13 F. Pereira, N. Tishby, and L. Lee. Distributional Clustering of English Words. In *Proceedings of the 31st Annual Meeting of the Association for Computational Linguistics*, pages 183–190, 1993.

14 P. H. A. Sneath and R. R. Sokal. *Numerical taxonomy: the principles and practice of numerical classification.* W. H. Freeman, 1973.

Measuring Learning Complexity with Criteria Epitomizers

John Case[1] **and Timo Kötzing**[2]

1 **Department of Computer and Information Sciences, University of Delaware, Newark, DE 19716-2586, USA**
case@cis.udel.edu

2 **Max-Planck Institute for Informatics, 66123 Saarbrücken, Germany**
koetzing@mpi-inf.mpg.de

Abstract

In prior papers, beginning with the seminal work by Freivalds et al. 1995, the notion of *intrinsic complexity* is used to analyze the learning complexity of sets of functions in a Gold-style learning setting. Herein are pointed out some weaknesses of this notion. Offered is an alternative based on *epitomizing sets* of functions – sets, which are learnable under a given learning criterion, but not under other criteria which are not at least as powerful.

To capture the idea of epitomizing sets, new reducibility notions are given based on *robust learning* (closure of learning under certain classes of operators). Various degrees of epitomizing sets are *characterized* as the sets complete with respect to corresponding reducibility notions! These characterizations also provide an easy method for showing sets to be epitomizers, and they are, then, employed to prove several sets to be epitomizing.

Furthermore, a scheme is provided to generate easily *very strong* epitomizers for a multitude of learning criteria. These strong epitomizers are so-called *self-learning* sets, previously applied by Case & Kötzing, 2010. These strong epitomizers can be generated and employed in a myriad of settings to witness the strict separation in learning power between the criteria so epitomized and other not as powerful criteria!

1998 ACM Subject Classification I.2.6 Learning

Keywords and phrases Algorithmic Learning Theory, Learning Complexity, Robustness in Learning

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.320

1 Introduction

We analyze the problem of algorithmically learning a description for an infinite sequence (a function from the natural numbers into the natural numbers) when presented larger and larger initial segments of that sequence. For example, a learner h might be presented more and more of the sequence $g = 1, 4, 9, 16, \ldots$. After each new datum of g, h may output a description of a function as its conjecture. For example, h might output a program for the constantly 1 function after seeing the first element of this sequence g and a program for the squaring function on all the other data from g. Many criteria for saying whether h is *successful* on g have been proposed in the literature. Gold, in his seminal paper [16], gave a first, simple learning criterion, later called in [11] *Ex-learning*[1], where a learner is

[1] *Ex* stands for *explanatory*.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 320–331
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

successful iff it eventually stops changing its conjectures, and its final conjecture is a correct description for 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 for which *sets of functions* $\mathcal{S}$ is there a *single learner* h learning *each* member of $\mathcal{S}$. We are interested herein in learning sets of total computable functions, and we will use (codes of) *programs* from a fixed *programming system* as possible conjectured descriptions for the functions.[2] This framework is known as *function learning in the limit* and has been studied extensively, using a wide range of learning criteria similar to Ex-learning (see, for example, the text book [19]).

Freivalds et al. [12] considered how to define *learning complexity* of a set of learnable functions. They introduced the seminal notion of *intrinsic complexity* and defined, for learning criteria I, a corresponding reducibility relation $\leq^I$. *Intrinsic* here is intrinsic to a learning task or problem $\mathcal{S}$, not to particular learning algorithms for $\mathcal{S}$. The idea is that, if $\mathcal{S} \leq^I \mathcal{S}'$, then $\mathcal{S}'$ is at least as hard to I-learn as is $\mathcal{S}$. In particular, [12] shows that, if $\mathcal{S} \leq^{\mathrm{Ex}} \mathcal{S}'$ and $\mathcal{S}'$ is Ex-learnable, then $\mathcal{S}$ is Ex-learnable. This intrinsic complexity has been further studied in some detail, see, for *example*, [12, 18, 17].

From [12], for a given learning criterion I, an I-learnable set of functions $\mathcal{S}_0$ is said to be $\leq^I$*-complete* iff, for all I-learnable sets of functions $\mathcal{S}$, $\mathcal{S} \leq^I \mathcal{S}_0$. As far as $\leq^I$ describes the relative difficulty of learnability, $\leq^I$-complete sets are the most difficult to I-learn. [12] shows that the set $\mathcal{S}_{\mathrm{FinSup}}$ of all computable functions of finite support[3] is $\leq^{\mathrm{Ex}}$-complete. These notions from [12] are structural analogs, for example, to the various notions from complexity theory of polynomial time reducibility and completeness.

There are at least two problems connected with the notion of intrinsic complexity from [12].

(i) For some learning criteria I, the relation $\leq^I$ is not very fine-grained. In particular, there are $\leq^I$-complete sets of functions which are also learnable with respect to much more restricted learning criteria (see Theorem 4.2 below).

(ii) There are learning criteria I and sets of functions $\mathcal{S}, \mathcal{S}'$ such that $\mathcal{S} \leq^I \mathcal{S}'$ and $\mathcal{S}'$ is I-learnable, but $\mathcal{S}$ is *not* I-learnable (see Theorem 4.3 below).

In this paper we quantify the difficulty of learning a given class of functions in a *new* way. First, we consider the following concept, essentially from [12]. A set of functions $\mathcal{S}$ *epitomizes* a learning criterion I *with respect to* some class of learning criteria $\mathcal{I}$, iff, $\mathcal{S}$ is I-learnable, and, for each $I' \in \mathcal{I}$, if some I-learnable task is too hard to be I'-learned, then $\mathcal{S}$ is already such an example task too hard to be I'-learned.[4]

We believe that epitomization nicely captures the learning complexity of a set of functions. Hence, the work herein aims at *finding* such epitomizers. Naturally, the interest is in epitomizers with respect to as large as possible classes of learning criteria $\mathcal{I}$. We give epitomizers with respect to classes of all learning criteria which are *robust* with respect to certain classes of operators (operating on functions). Essentially, a learning criterion I is *robust* with respect to a given class of operators $\mathfrak{O}$ iff, for each I-learnable task $\mathcal{S}$ and each

[2] One could, for example, think of the programming system as one of Java, C, Turing machines, ...

[3] A (total) function has *finite support* iff only finitely many arguments have a function value other than 0.

[4] Note that [12] called epitomizing sets *characteristic*. To our best knowledge, neither this term nor the concept caught on in the later literature — until now.

operator $\Theta \in \mathfrak{O}$, $\Theta(\mathcal{S})$ is also I-learnable, i.e., the class of I-learnable sets of functions is closed under operators from $\mathfrak{O}$.[5]

Furthermore, for any set of operators $\mathfrak{O}$, we define a reducibility $\leq^{\mathfrak{O}}$ and a corresponding completeness notion. As an important first theorem we have that a set $\mathcal{S}$ epitomizes a learning criterion I with respect to all $\mathfrak{O}$-robust learning criteria iff $\mathcal{S}$ is $\leq^{\mathfrak{O}}$-complete for all I-learnable sets (Theorem 3.6 below)! The benefits of this theorem are twofold.

First, since, as noted above, we believe that epitomization captures the complexity of learning, this Theorem 3.6 entails that our reducibility notions also capture this complexity.

Secondly, we now need only to prove completeness to get epitomization!

Other than structural insight, we get the following two benefits from epitomizers.

First, we can use epitomizers to show the identity of learning power of two learning criteria. For example, Theorem 5.2 establishes $\mathcal{S}_{\text{FinSup}}$ to be epitomizing with respect to various learning criteria and sets of operators. In Corollary 5.3 below we use this to show a learning criterion I to be as powerful as one of the epitomized learning criteria by showing that $\mathcal{S}_{\text{FinSup}}$ can be I-learned. With classic methods, the proof of this result might have required tedious work with attention to detail, while we can conclude it as a corollary to *structural properties* uncovered by our theorems.

The second way in which epitomization helps us is by providing canonical candidates to witness the separation of two learning criteria. To this end, *self-learning* classes (see Theorem 5.7) are particularly useful epitomizers and are used in the literature to prove particularly difficult separations (see, for example, [9], which solves two previously open problems using this technique, and see also [10]).

[12] noted that their $\leq^{\text{Ex}}$-completeness does not give epitomization with respect even to the set of learning criteria considered in [12].

Thus, we believe our approach to complexity of learning is both more *comprehensive* and more *useful* than the notion of intrinsic complexity from [12].

We present mathematical preliminaries in Section 2. The notions discussed above and some first theorems about them are given in Section 3, including the mentioned important characterization of epitomizers as complete sets (Theorem 3.6 below).

Section 4 gives definitions and results regarding the notion of intrinsic complexity introduced in [12]. We already mentioned our Theorems 4.2 and 4.3 below which witness drawbacks of this older notion; furthermore, in Theorem 4.5 below, we characterize $\leq^{\text{Ex}}$ in terms of one of our reducibility notions, and conclude in Corollary 4.6 that all sets complete with respect to a central one of our reducibility notions are $\leq^{\text{Ex}}$-complete.

Finally, in Section 5, we present a series of tasks and state which learning criteria they epitomize *at what strength.* N.B. Epitomizers with respect to larger classes of learning criteria are stronger. As indicated above, we give each epitomization result, for some set of operators $\mathfrak{O}$, and with respect to the corresponding set of $\mathfrak{O}$-robust learning criteria. N.B. It will be seen that the smaller the set of operators $\mathfrak{O}$, the larger the set of $\mathfrak{O}$-robust learning criteria. Theorem 5.2 entails that $\mathcal{S}_{\text{FinSup}}$, the set of functions of finite support introduced above, is a very weak epitomizer. Some so called self-*describing* classes are also surprisingly weak epitomizers (Theorem 5.4 below); some are of considerably greater strength (Theorem 5.5

[5] In the previous literature, a set $\mathcal{S}$ was called *robustly I-learnable* iff, for *all recursive operators* [22] Θ, the class of total functions in $\Theta(\mathcal{S})$ is I-learnable (see, for example, [15]). The motivation for such past notions of robustness was to eliminate self-referential examples. The motivation herein is quite different. Herein, as will be seen, it is very interesting that which operators are to be considered can be restricted, and ask for *all* I-learnable tasks to be robust with respect to some possibly restricted class of operators.

below); but, interestingly, the strongest are self-*learning* sets (Theorem 5.7 below).

Some of our proofs involve subtle infinitary program self-reference arguments employing (variants of) the Operator Recursion Theorem (ORT) from [5, 6, 19]. Note that, due to space constraints, all proofs are omitted from the paper.

We are working on extending the present paper to employ self-learning sets which work for learning criteria, unlike those herein, which require the learners to be total on *all* inputs.

2 Mathematical Preliminaries

Unintroduced computability-theoretic notions follow [22].

$\mathbb{N}$ denotes the set of natural numbers, $\{0, 1, 2, \ldots\}$.

The symbols $\subseteq, \subset, \supseteq, \supset$ respectively denote the subset, proper subset, superset and proper superset relation between sets. The symbol $\setminus$ denotes set-difference.

The quantifier $\forall^\infty x$ means "for all but finitely many $x \in \mathbb{N}$"; the quantifier $\exists^\infty x$ means "for infinitely many $x \in \mathbb{N}$". For any set A, $\mathrm{card}(A)$ denotes its cardinality, $\mathrm{Pow}(A)$ denotes the set of all subsets of A.

With $\mathfrak{P}$ and $\mathfrak{R}$ we denote, respectively, the set of all partial and of all total functions $\mathbb{N} \rightarrow \mathbb{N}$. With dom and range we denote, respectively, domain and range of a given function. Set-theoretically, (partial) functions are identified with their graphs, i.e., they are treated as sets of ordered pairs, and we sometimes compare them by $\subseteq$.

We sometimes denote a partial function f of $n > 0$ arguments $x_1, \ldots, x_n$ in lambda notation (as in Lisp) as $\lambda x_1, \ldots, x_n \centerdot f(x_1, \ldots, x_n)$. For example, with $c \in \mathbb{N}$, $\lambda x \centerdot c$ is the constantly c function of one argument.

If $f \in \mathfrak{P}$ 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 $f(x)\!\downarrow = p$.

We say that $f \in \mathfrak{P}$ *converges to* p iff $\forall^\infty x : f(x)\!\downarrow = p$; we write $f \rightarrow p$ to denote this.[6]

For any (possibly partial) predicate P, we let $\mu x \centerdot P(x)$ denote the least x such that $P(x)$ and, for all $y < x$, $P(x)\!\downarrow$ (if no such x exists, $\mu x \centerdot P(x)$ is undefined).

We fix any computable 1-1 and onto pairing function $\langle \cdot, \cdot \rangle : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$.[7] With π_1 and π_2, respectively, we denote decoding into first and second arguments of pairing, respectively. Whenever we consider tuples of natural numbers as input to $f \in \mathfrak{P}$, it is understood that the general coding function $\langle \cdot, \cdot \rangle$ is used to (left-associatively) code the tuples into a single natural number (but we will not necessarily state the pairing explicitly).

For any $g \in \mathfrak{P}$ and $x \in \mathbb{N}$, we let $g[x]$ denote the sequence of the numbers $g(0), \ldots, g(x - 1)$, if all are defined, and $\uparrow$ otherwise.

A partial function $f \in \mathfrak{P}$ is *partial computable* iff there is a deterministic, multi-tape Turing machine which, on input x, returns $f(x)$ if $f(x)\!\downarrow$, and loops infinitely if $f(x)\!\uparrow$. $\mathcal{P}$ and $\mathcal{R}$ denote, respectively, the set of all partial computable and the set of all (total) computable functions $\mathbb{N} \rightarrow \mathbb{N}$. The functions in $\mathcal{R}$ are called *computable functions*.

We let φ be any fixed acceptable programming system for the partial computable functions $\mathbb{N} \rightarrow \mathbb{N}$ with associated complexity measure Φ [19]. Further, we let φ_p denote the partial computable function computed by the φ-program with code number p, and we let Φ_p denote the partial computable *complexity* function of the φ-program with code number p.

[6] $f(x)$ converges should not be confused with f converges *to*.

[7] For a linear-time computable and invertible example, see [23, Section 2.3].

Whenever we consider sequences or finite sets as input to functions, we assume these objects to be appropriately coded as natural numbers. Similarly, when functions are defined to give non-numeric output, for example, when the outputs are in $\mathbb{N} \cup \{?\}$, we implicitly assume $\mathbb{N} \cup \{?\}$ to be appropriately coded onto the natural numbers.

We use complexity theoretic notions as introduced in [23]. We let **LinF** be the set of all linear time computable functions. A function g is called *linlin* iff g is computable in linear time and there is a linear time computable function g^{-1} such that $g^{-1} \circ g = \lambda x \centerdot x$. We let **LL** be the set of all linlin functions.

Learning in the Limit

A *learner* is a partial computable function. A *target* is a total computable function g; a *learning task* is a set of targets $\mathcal{S} \subseteq \mathcal{R}$.

A learning criterion consists of three parts which, together, determine whether a given learner is successful on a given learning task.

Firstly, the learning criterion has to specify what learners are allowed. This is called a *learner admissibility restriction*, and is modeled as a set $\mathcal{C} \subseteq \mathcal{P}$, the set of all admissible learners.

Secondly, the learning criterion has to specify how learner and target interact. This part is modeled as a *sequence generating operator*, which is an operator β taking as arguments a learner h and a target g and that outputs a function p. We call p the *learning sequence* of h given g. For this paper, we think of p as the sequence of conjectured programs of h on g.

Thirdly, the learning criterion has to specify which learning sequences are to be considered "successful" on a given target. This is done with a *sequence acceptance criterion*, a total binary predicate δ on a learning sequence and a target function.[8]

For $\mathcal{C}$ a learner admissibility restriction, β a sequence generating operator, δ a sequence acceptance criterion and h a learner, we call $(\mathcal{C}, \beta, \delta)$ a learning criterion. For every learning criterion I with $I = (\mathcal{C}, \beta, \delta)$ we let $\mathcal{C}_I = \mathcal{C}$, $\beta_I = \beta$ and $\delta_I = \delta$. Let $I = (\mathcal{C}, \beta, \delta)$ be a learning criterion. We proceed by giving definitions for I-learning.

We say that *h I-learns a learning task $\mathcal{S}$* iff, $h \in \mathcal{C}$ and, for all $g \in \mathcal{S}$, with $p = \beta(h, g)$, $(p, g) \in \delta$. We denote by $\mathfrak{S}(I)$ and also by $\mathcal{C}\beta\delta$ the set of all I-learnable learning tasks.[9] With an abuse of notation, we sometimes also use $\mathcal{C}\beta\delta$ to denote I.

Any set of complexity-bounded functions is an example learner admissibility restriction, as are $\mathcal{R}$ and $\mathcal{P}$. We omit mentioning $\mathcal{C}$, if $\mathcal{C} = \mathcal{P}$ (no restriction on the learner).

We give the following three examples for sequence generating operators. Let $\mathbf{G}$ be defined thus.[10]

$$\forall h, g, i : \mathbf{G}(h, g)(i) = h(g[i]). \tag{1}$$

Let $\mathbf{It}$ be defined thus.[11]

$$\forall h, g : \mathbf{It}(h, g)(0) = ? \ \wedge \ \forall i : \mathbf{It}(h, g)(i + 1) = h(\mathbf{It}(h, g)(i), \langle i, g(i) \rangle). \tag{2}$$

Lastly, we give *transductive* learning [7].

$$\forall h, g : \mathbf{Td}(h, g)(0) = ? \ \wedge \ \forall i : \mathbf{Td}(h, g)(i + 1) = h(\langle i, g(i) \rangle). \tag{3}$$

[8] Herein, our βs and δs can essentially be modeled as multi-argument variants of Rogers' [22] *recursive operators*.

[9] Note that "$\mathcal{C}\beta\delta$" is the classical way to denote this, while we use "$\mathfrak{S}(I)$" whenever $\mathcal{C}$, β and δ are not explicit.

[10] $\mathbf{G}$ stands for Gold identification [16].

[11] $\mathbf{It}$ stands for iterative [14, 24].

For sequence acceptance criteria, we give the following five examples. We define *explanatory* learning [16] as follows.

$$\mathbf{Ex} = \{(p, g) \mid p \text{ total and } \exists e : p \to e \text{ and } \varphi_e = g\}.$$

Finite learning is given by

$$\mathbf{Fin} = \{(p, g) \mid p \text{ total and } \exists e \in \mathbb{N} : e \in \mathrm{range}(p) \subseteq \{e, ?\} \text{ and } \varphi_e = g\}.$$

Further, we define a sequence acceptance criterion corresponding to *postdictively complete* learning[12] [2, 4, 24].

$$\mathbf{Pcp} = \{(p, g) \mid p \text{ total and } \forall i : g[i] \subseteq \varphi_{p(i+1)}\}.$$

For *conservative* learning [1] we give the following sequence acceptance criterion.

$$\mathbf{Conv} = \{(p, g) \mid p \text{ total and } \forall i : p(i+1) \neq p(i) \Rightarrow g[i] \not\subseteq \varphi_{p(i)}\}.$$

Finally, behaviorally correct learning as given by [3, 11] is associated with

$$\mathbf{Bc} = \{(p, g) \mid p \text{ total and } \forall^\infty i : \varphi_{p(i)} = g\}.$$

Any two sequence acceptance criteria δ and δ' can be combined by intersecting them. For ease of notation we write $\delta\delta'$ instead of $\delta \cap \delta'$.

For more examples of the above concepts, see [20].

3 Concept Definitions

In this section, we give the key concepts used in this paper in Definition 3.3. Before we can get to that, we define pre-orders and associated notions, as well as several sets of operators, for which we give examples and remark on some easy properties.

The main theorem of this section is Theorem 3.6, which shows the important connections between complete sets and epitomizers.

Let S be a set and $\preceq$ a binary relation on that set. We call $\preceq$ a *pre-order* iff $\preceq$ is reflexive and transitive. For $s \in S$ and $T \subseteq S$, we say that s is *$\preceq$-complete for T* iff, $s \in T$ and, for all $t \in T$, $t \preceq s$.

Next we define several sets of operators. For illustration, see Example 3.2.

▶ **Definition 3.1.** A function $\Theta : \mathcal{P} \to \mathcal{P}$ is called an *operator*. We define the following sets of operators.

- Let $\mathfrak{O}_{\text{eff}}$ be the set of all effective operators [22], i.e., all operators Θ such that there is a computable function $s \in \mathcal{R}$ with $\forall e : \Theta(\varphi_e) = \varphi_{s(e)}$.[13]
- Let $\mathcal{C} \subseteq \mathcal{P}$. Let $\mathfrak{O}_{\text{loc}}^{\mathcal{C}}$ be the set of all $\Theta \in \mathfrak{O}_{\text{eff}}$ such that, for all g, x, $\Theta(g)[x]$ depends only on $g[x]$, i.e., if there is a function f with $\forall g \in \mathcal{P}, \forall x : \Theta(g)[x] = f(g[x])$, and $f \in \mathcal{C}$. Note that $\Theta \in \mathfrak{O}_{\text{loc}}$ is equivalent to

$$\forall g, g' \in \mathcal{P} \forall x : g[x] = g'[x] \Rightarrow \Theta(g)[x] = \Theta(g')[x].^{14} \qquad (4)$$

[12] Also called *consistent* learning.
[13] Note that, without loss of generality, we can take s to be linlin.

- Let $\mathcal{C} \subseteq \mathcal{P}$. Let $\mathfrak{O}^{\mathcal{C}}_{\text{elemWise}}$ be the set of all $\Theta \in \mathfrak{O}_{\text{eff}}$ such that there is a function $f \in \mathcal{C}$ such that $\forall g \in \mathcal{P} : \Theta(g) = \lambda x \centerdot f(g(x), x)$.[15] We write $\mathfrak{O}_{\text{elemWise}}$ for $\mathfrak{O}^{\mathcal{P}}_{\text{elemWise}}$.

Next, we define sets of *left-invertible* operators. For any set of operators $\mathfrak{O}$ and $\mathcal{S} \subseteq \mathcal{R}$, we let $\text{LInv}(\mathfrak{O}; \mathcal{S}) = \{\Theta \in \mathfrak{O} \mid \Theta(\mathcal{S}) \subseteq \mathcal{R} \wedge \exists \hat{\Theta} \in \mathfrak{O}\ \forall g \in \mathcal{S} : (\hat{\Theta} \circ \Theta)(g) = g\}$.

Clearly, $\mathfrak{O}_{\text{elemWise}} \subset \mathfrak{O}_{\text{loc}} \subset \mathfrak{O}_{\text{eff}}$.

▶ **Example 3.2.** For illustration, we give the following example operators.

- The operator Θ such that $\forall g \in \mathcal{P} \forall x : \Theta(g)(x) = g(x + 1)$ is in $\mathfrak{O}_{\text{eff}}$, but not in $\mathfrak{O}_{\text{loc}}$ or $\mathfrak{O}_{\text{elemWise}}$; furthermore, Θ is not in $\text{LInv}(\mathfrak{O}_{\text{eff}})$ (Θ is not 1-1, hence cannot be left-inverted).
- The operator Θ such that

$$\forall g \in \mathcal{P} \forall x : \Theta(g)(x) = \begin{cases} 0, & \text{if } x = 0; \\ g(x-1), & \text{otherwise,} \end{cases} \tag{5}$$

  is in $\mathfrak{O}_{\text{eff}}$ and in $\mathfrak{O}_{\text{loc}}$, but not in $\mathfrak{O}_{\text{elemWise}}$; furthermore, Θ is in $\text{LInv}(\mathfrak{O}_{\text{eff}})$, but not in $\text{LInv}(\mathfrak{O}_{\text{loc}})$ (Θ has a computable left-inverse, but not a local one).
- The operator Θ such that $\forall g \in \mathcal{P} \forall x : \Theta(g)(x) = x + g(x)$ is in $\mathfrak{O}_{\text{eff}}$, $\mathfrak{O}_{\text{loc}}$ and also $\mathfrak{O}_{\text{elemWise}}$; furthermore, Θ is even in $\text{LInv}(\mathfrak{O}_{\text{elemWise}})$.

Now we give the definition of the central notions of this paper.

▶ **Definition 3.3.** Let $\mathfrak{O}$ be a set of operators and I a learning criterion. Let $\mathcal{S}, \mathcal{S}_0, \mathcal{S}_1 \subseteq \mathcal{R}$.

(i) I is called *$\mathfrak{O}$-robust* iff, for all $\mathcal{S} \subseteq \mathcal{R}$ and $\Theta \in \text{LInv}(\mathfrak{O}; \mathcal{S})$,

$$\mathcal{S} \in \mathfrak{S}(I) \Rightarrow \Theta(\mathcal{S}) \in \mathfrak{S}(I).^{16} \tag{6}$$

(ii) We say that $\mathcal{S}$ *epitomizes* I with respect to a set of learning criteria $\mathcal{I}$, iff $\mathcal{S} \in \mathfrak{S}(I)$ and

$$\forall I' \in \mathcal{I} : [\mathcal{S} \in \mathfrak{S}(I') \Leftrightarrow \mathfrak{S}(I) \subseteq \mathfrak{S}(I')]\,.^{17} \tag{7}$$

(iii) If $\mathcal{S}$ epitomizes I with respect to the set of all $\mathfrak{O}$-robust learning criteria, then we say $\mathcal{S}$ *$\mathfrak{O}$-epitomizes I*.

(iv) We say that $\mathcal{S}$ *$\mathfrak{O}$-generates* I iff $\{\Theta(\mathcal{S}') \mid \mathcal{S}' \subseteq \mathcal{S}, \Theta \in \text{LInv}(\mathfrak{O}; \mathcal{S}')\}$ is the set of all I-learnable functions.

(v) $\mathcal{S}_0 \leq^{\mathfrak{O}} \mathcal{S}_1$ iff there is an operator $\Theta \in \text{LInv}(\mathfrak{O}; \mathcal{S}_0)$ such that $\Theta(\mathcal{S}_0) \subseteq \mathcal{S}_1$.

As an interesting first observation on epitomizers, we make the following remark regarding separations from unions of learning criteria.

▶ **Proposition 3.4.** Let I be a learning criterion and $\mathcal{I}$ a set of learning criteria with $\forall I' \in \mathcal{I} : \mathfrak{S}(I) \setminus \mathfrak{S}(I') \neq \emptyset$. Suppose there is a set epitomizing I with respect to $\mathcal{I}$. Then

$$\mathfrak{S}(I) \setminus \bigcup_{I' \in \mathcal{I}} \mathfrak{S}(I') \neq \emptyset.$$

[15] We call these operators *element wise*.

Theorem 5.7 provides very strong existence results for epitomizers which can be used to satisfy the corresponding hypothesis of Proposition 3.4.[18]

We can use Proposition 3.4 to give cases where epitomizers do not exist: For example, let I be reliable learnability and let $\mathcal{I}$ be the set of all learning criteria of delayed postdictive completeness; then $\forall I' \in \mathcal{I} : \mathfrak{S}(I) \setminus \mathfrak{S}(I') \neq \emptyset$ and $\mathfrak{S}(I) = \bigcup_{I' \in \mathcal{I}} \mathfrak{S}(I')$, which shows that there is no epitomizer of I with respect to $\mathcal{I}$ (see [8] for the definitions; the result will be included in an extension of [8]).[19]

The following theorem gives a number of general observations regarding the concepts introduced in Definition 3.3.

▶ **Proposition 3.5.** Let $\mathfrak{O}, \mathfrak{O}'$ be sets of operators, I a learning criterion.

(i) Let I be $\mathfrak{O}$-robust. Then, for all $\mathcal{S} \subseteq \mathcal{R}$, $\Theta \in \mathrm{LInv}(\mathfrak{O}; \mathcal{S})$ with $\Theta(\mathcal{S}) \subseteq \mathcal{R}$,

$$\mathcal{S} \in \mathfrak{S}(I) \Leftrightarrow \Theta(\mathcal{S}) \in \mathfrak{S}(I).$$

(ii) If $\mathfrak{O} \in \{\mathfrak{O}_{\text{eff}}, \mathfrak{O}_{\text{loc}}, \mathfrak{O}_{\text{elemWise}}\}$, then $\leq^{\mathfrak{O}}$ is a pre-order.

(iii) Suppose I is $\mathfrak{O}$-robust. Suppose $\mathcal{S} \in \mathfrak{S}(I)$. Then, for all $\mathcal{S}' \subseteq \mathcal{R}$,

$$\mathcal{S}' \leq^{\mathfrak{O}} \mathcal{S} \Rightarrow \mathcal{S}' \in \mathfrak{S}(I).$$

Next is the central theorem of this section, showing that, for important sets of operators $\mathfrak{O}$ and certain learning criteria I, $\leq^{\mathfrak{O}}$-completeness for I *characterizes* $\mathfrak{O}$-epitomization of I.

▶ **Theorem 3.6.** Let $\mathfrak{O}$ be a set of operators containing the identity and which is closed under composition. Let I be a $\mathfrak{O}$-robust learning criterion. Let $\mathcal{S} \subseteq \mathcal{R}$. The following are equivalent.

(i) $\mathcal{S}$ $\mathfrak{O}$-epitomizes I.
(ii) $\mathcal{S}$ $\mathfrak{O}$-generates I.
(iii) $\mathcal{S}$ is $\leq^{\mathfrak{O}}$-complete for I.

4 Connection to Intrinsic Complexity

We will now give the definitions of intrinsic complexity. Some version thereof was introduced in [12], here we give an interpretation that fits our formalism. After that we will give theorems regarding the shortcomings of intrinsic complexity, as discussed in the introduction.

In particular, Theorem 4.2 gives an example $\leq^{\mathbf{GEx}}$-complete set of functions which is nonetheless learnable in much more restricted criteria. Then, in Theorem 4.3, we give two natural learning criteria I for which the learnable sets are not downward closed with respect to $\leq^{I}$. For the two criteria, the cause of this failure of closure is different: in one case it is a local restriction on the conjectures, in the other a memory restriction on the learner.

Finally, we show the equivalence of $\leq^{\mathbf{GEx}}$ with one of our reducibility notions in Theorem 4.5.

[18] We give the following example for illustration. Let, for all $a \in \mathbb{N}$, $\mathbf{Ex}^a$ be like $\mathbf{Ex}$, but, for success, the final program is allowed to be incorrect on up to a places. Further, let $\mathbf{Ex}^*$ be like $\mathbf{Ex}$, but the final program is allowed to be incorrect on up to finitely many places. From [11] we know that, for all $a \in \mathbb{N}$, $\mathbf{GEx}^a \subset \mathbf{GEx}^{a+1}$; hence, $\mathbf{GEx}^a \subset \mathbf{GEx}^*$. Theorem 5.7 provides an appropriate epitomizer for $\mathbf{GEx}^*$, so that we can deduce, with Proposition 3.4, $\bigcup_{a \in \mathbb{N}} \mathbf{GEx}^a \subset \mathbf{GEx}^*$ (which was shown in [11]).

[19] We are thankful to an anonymous referee who pointed out a (different) example for the nonexistence of epitomizers.

▶ **Definition 4.1.** Let $I = (\mathcal{C}, \beta, \delta)$ be a learning criterion, and g a function. We say that a sequence p is *I-admissible* for g iff $(p, g) \in \delta$.[20]

Let $\mathcal{S}_0, \mathcal{S}_1 \subseteq \mathcal{R}$, and an identification criterion I be given. We say that $\mathcal{S}_0 \leq^I \mathcal{S}_1$ iff there exist recursive operators Θ and Ψ such that, for any function $g \in \mathcal{S}_0$,

(i) $\Theta(g) \in \mathcal{S}_1$, and
(ii) for any I-admissible sequence p for $\Theta(g)$, $\Psi(p)$ is an I-admissible sequence for g.

We say that a set $\mathcal{S} \subseteq \mathcal{R}$ is $\leq^I$-complete, iff $\mathcal{S}$ is $\leq^I$-complete for $\mathfrak{S}(I)$.

▶ **Theorem 4.2** ([12] & [3, 4])**.** $\mathcal{S}_{\mathrm{FinSup}}$ is $\leq^{\mathbf{GEx}}$-complete, but $\mathcal{S}_{\mathrm{FinSup}} \in \mathbf{GPcpEx} \subset \mathbf{GEx}$.

The following theorem shows two criteria classes to be not closed downwards with respect to their respective intrinsic reducibility notions.

▶ **Theorem 4.3.**

(i) There are sets $\mathcal{S}_0$ and $\mathcal{S}_1 \subseteq \mathcal{R}$ such that $\mathcal{S}_0 \leq^{\mathbf{ItEx}} \mathcal{S}_1$ and $\mathcal{S}_1 \in \mathbf{ItEx}$, but $\mathcal{S}_0 \notin \mathbf{ItEx}$.
(ii) There are sets $\mathcal{S}_0$ and $\mathcal{S}_1 \subseteq \mathcal{R}$ such that $\mathcal{S}_0 \leq^{\mathbf{GPcpEx}} \mathcal{S}_1$ and $\mathcal{S}_1 \in \mathbf{GPcpEx}$, but $\mathcal{S}_0 \notin \mathbf{GPcpEx}$.

In order to characterize the reducibility of intrinsic complexity in terms of our notions, we give the following definitions, extending notions from Section 3.

▶ **Definition 4.4.** Let $\mathfrak{O}, \mathfrak{O}'$ be sets of operators and $\mathcal{S} \subseteq \mathcal{R}$.

(i) Let $\mathrm{LInv}(\mathfrak{O}, \mathfrak{O}'; \mathcal{S}) = \{\Theta \in \mathfrak{O} \mid \exists \hat{\Theta} \in \mathfrak{O}' \; \forall g \in \mathcal{S} : (\hat{\Theta} \circ \Theta)(g) = g\}$.
(ii) For two sets $\mathcal{S}_0, \mathcal{S}_1 \subseteq \mathcal{R}$, we write $\mathcal{S}_0 \leq^{(\mathfrak{O}, \mathfrak{O}')} \mathcal{S}_1$ iff there is an operator $\Theta \in \mathrm{LInv}(\mathfrak{O}, \mathfrak{O}'; \mathcal{S}_0)$ such that $\Theta(\mathcal{S}_0) \subseteq \mathcal{S}_1$.
(iii) Furthermore, let $\mathfrak{O}_{\mathrm{limPEff}}$ be the set of all partial operators Θ such that there is a computable function $s \in \mathcal{R}$ with

$$\forall e : \Theta(\varphi_e) = \begin{cases} \varphi_p, & \text{if } \lambda t \centerdot s(e,t) \text{ converges to some } p; \\ \uparrow, & \text{otherwise.}^{21} \end{cases} \tag{8}$$

Now we show the equivalence of one particular reducibility notion of intrinsic complexity with one of our extended variants from Definition 4.4. Note that a similar characterization can be made for **GBc**.

▶ **Theorem 4.5.** We have

$$\leq^{(\mathfrak{O}_{\mathrm{eff}}, \mathfrak{O}_{\mathrm{limPEff}})} \quad \text{equals} \quad \leq^{\mathbf{GEx}} .$$

We get the following Corollary from Theorem 4.5.

▶ **Corollary 4.6.** Let I be a learning criterion with $\delta_I = \mathbf{Ex}$ as sequence acceptance criterion. We have that $\leq^{\mathfrak{O}_{\mathrm{eff}}}$ is a subrelation of $\leq^{\mathbf{GEx}}$; in particular, for all $\mathcal{S} \subseteq \mathcal{R}$

$$\mathcal{S} \text{ is } \leq^{\mathfrak{O}_{\mathrm{eff}}}\text{-complete for } \mathfrak{S}(I) \Rightarrow \mathcal{S} \text{ is } \leq^I\text{-complete.}$$

[20] I-admissibility is not to be confused with learner admissibility restrictions.
[21] Note that the operators from $\mathfrak{O}_{\mathrm{limPEff}}$ resemble those from [21] and [13].

5 Specific Function Learning Epitomizers

In this section we present several epitomizers, starting with the very natural class $\mathcal{S}_{\mathrm{FinSup}}$ in Theorem 5.2. After noting an immediate corollary, we show a self-*describing* class to be an epitomizer. Finally, we give a construction for self-*learning* classes and show them to be very powerful epitomizers in Theorem 5.7, i.e., epitomizing with respect to a very wide range of learning criteria, much wider than for self-descibing classes (which is in turn wider than for $\mathcal{S}_{\mathrm{FinSup}}$).

Note that all proofs rely implicitly on Theorem 3.6, as they show completeness and derive epitomization from that.

But first, we give a table of examples of which learning criteria are robust with respect to which set of operators.

▶ **Example 5.1.** Let $\mathcal{C} \subseteq \mathcal{P}$ contain all linlin functions. Let $\mathcal{F} \subseteq \mathcal{P}$ be closed under $\mathcal{C}$-composition and contain all linear time computable functions.

The following table states several kinds of robustness of learning criteria with respect to certain sets of operators.

$\mathfrak{O}_{\mathrm{eff}}$	$\mathcal{F}\mathbf{GEx}$, $\mathcal{F}\mathbf{GBc}$, $\mathcal{F}\mathbf{GFin}$
$\mathfrak{O}^{\mathcal{C}}_{\mathrm{loc}}$	$\mathcal{F}\mathbf{GPcpEx}$, $\mathcal{F}\mathbf{GConvEx}$, $\mathcal{F}\mathbf{GPcpConvEx}$
$\mathfrak{O}^{\mathcal{C}}_{\mathrm{elemWise}}$	$\mathcal{F}\mathbf{ItEx}$, $\mathcal{F}\mathbf{ItConvEx}$, $\mathcal{F}\mathbf{ItPcpConvEx}$, $\mathcal{F}\mathbf{TdEx}$, $\mathcal{F}\mathbf{TdBc}$

Note that each criterion in any given row just above could also be listed in any lower row.

Next, we show $\mathcal{S}_{\mathrm{FinSup}}$ to epitomize various *particular* learning criteria with respect to various *sets* of learning criteria.

▶ **Theorem 5.2.** We have

(i) $\mathcal{S}_{\mathrm{FinSup}}$ $\mathfrak{O}_{\mathrm{eff}}$-epitomizes $\mathbf{GEx}$;
(ii) $\mathcal{S}_{\mathrm{FinSup}}$ does *not* $\mathfrak{O}_{\mathrm{loc}}$-epitomize $\mathbf{GEx}$;
(iii) $\mathcal{S}_{\mathrm{FinSup}}$ $\mathfrak{O}_{\mathrm{loc}}$-epitomizes $\mathbf{GPcpEx}$.
(iv) $\mathcal{S}_{\mathrm{FinSup}}$ does *not* $\mathfrak{O}_{\mathrm{elemWise}}$-epitomize $\mathbf{GPcpEx}$;

From Theorem 5.2 we can directly get the following corollary, showing an identity of learning criteria power. The very short proof shows the power of our notions of epitomizers and robustness.

▶ **Corollary 5.3.** We have

$$\mathbf{GPcpEx} \quad = \quad \mathbf{GPcpConvEx}.$$

Next we analyze a set of self-describing functions which was first given in [11] and used a lot in [19]. Theorem 5.4 shows that this set is not a very good epitomizer.

▶ **Theorem 5.4.** Let $\mathcal{S}_0 = \{g \in \mathcal{R} \mid \varphi_{g(0)} = g\}$. Then $\mathcal{S}_0$ $\mathfrak{O}_{\mathrm{eff}}$-epitomizes $\mathbf{GFin}$. However, $\mathcal{S}_0$ does not $\mathfrak{O}_{\mathrm{loc}}$-epitomize any learning criterion that can be built from components given in this paper as $\mathcal{S}_0 \equiv^{\mathfrak{O}_{\mathrm{loc}}} \{g \in \mathcal{R} \mid \forall x > 0 : g(x) = 0\}$.

Next we analyze a set of self-describing functions which was used in [11] to show the separation of $\mathbf{GBc}$ and (a stronger version of) $\mathbf{GEx}$. Theorem 5.5 shows that this set necessarily shows the separation of $\mathbf{GEx}$ and $\mathbf{GBc}$, if any set does.

▶ **Theorem 5.5.** Let $\mathcal{S}_0 = \{g \in \mathcal{R} \mid \forall^\infty x : \varphi_{g(x)} = g\}$. Then $\mathcal{S}_0$ $\mathfrak{O}^{\mathbf{LL}}_{\text{loc}}$-epitomizes **GBc**. However, $\mathcal{S}_0$ does not $\mathfrak{O}_{\text{elemWise}}$-epitomize **GBc**, as $\mathcal{S}_{\text{FinSup}} \not\leq^{\mathfrak{O}_{\text{elemWise}}} \mathcal{S}_0$.[22]

Theorem 5.7 gives examples for *self-learning* classes of functions, which turn out to be very strong epitomizers. In order to define these classes, we need the following notions of *computable robustness* and *data normality.*

▶ **Definition 5.6.** Let I be a learning criterion.

We call I *computably element-wise robust* iff I is element-wise robust in a constructive way, i.e., there is an effective operator $\Psi \in \mathfrak{O}_{\text{eff}}$ such that, for all $h, e \in \mathcal{P}$, $e \circ I(h) \subseteq I(\Psi(h, e))$.

We call I *data normal* iff (a) – (c) below.

(a) There is $f_I \in \mathcal{R}$ such that

$$\forall h \in \mathcal{P} \forall g \in \mathcal{R} \forall i : \beta_I(h, g)(i) = h(f_I(g[i], \beta_I(h, g)[i])).^{23} \tag{9}$$

(b) There is a function $d_I \in \mathcal{R}$ such that

$$\forall h \in \mathcal{P} \forall g \in \mathcal{R} \forall i : d_I(f_I(g[i], \beta_I(h, g)[i])) = \begin{cases} ?, & \text{if } i = 0; \\ g(i-1), & \text{otherwise.}^{24} \end{cases} \tag{10}$$

(c) There is an effective operator $\hat{\Psi} \in \mathfrak{O}_{\text{eff}}$ such that, for all $h \in \mathcal{P}$, $I(h) \subseteq I(\hat{\Psi}(h))$ and $\hat{\Psi}(h)(f_I(\emptyset, \emptyset)) = ?$.[25]

▶ **Theorem 5.7.** Let I be a computably element-wise robust learning criterion with $\mathcal{C}_I = \mathcal{P}$ (i.e., I does not impose global restrictions on the learner). Suppose I is data normal as witnessed by f and d. Let h_0 be such that

$$\forall x : h_0(x) = \begin{cases} ?, & \text{if } d(x) = ?; \\ \varphi_{d(x)}(x), & \text{otherwise.} \end{cases} \tag{11}$$

Further, let $\mathcal{S}_0 = I(h_0)$.[26] Then $\mathcal{S}_0$ $\mathfrak{O}^{\mathbf{LL}}_{\text{elemWise}}$-epitomizes I.

Theorem 5.7 provides epitomizing sets for the learning criteria $\beta\delta$ with β and δ as explicitly given in this paper, and many more. Furthermore, $\mathcal{S}_0$ of Theorem 5.7 epitomizes with respect to all learning criteria that can be built from the example components given in this paper (including the ones with learner admissibility restrictions), as they are all $\mathfrak{O}^{\mathbf{LL}}_{\text{elemWise}}$-robust. In particular, Theorem 5.7 provides a superior epitomizer for **GBc** than the epitomizer of Theorem 5.5.[27]

[22] This uses Theorem 3.6.

[24] Intuitively, in the ith round the learner has access to the $(i-1)$st data item.

[25] Intuitively, without loss of generality, the output based on no data of a learner equals ?.

[26] $\mathcal{S}_0$ is a *self-learning class*. Roughly, the learner (h_0 in Theorem 5.7) defining such a set ($\mathcal{S}_0$ in Theorem 5.7) just *runs* each input datum as coding a program to determine its corresponding conjecture to output [9, 10].
Note that h_0 is not total, hence, not polynomial time computable, etc. It is work in progress to extend self-learning classes for criteria separations to cases which cover associated learners interestingly more restricted than simply being partial computable, e.g., restricted to being linear time computable.

[27] The first author of the present paper, when he was co-creating [11], had the intuition that, for any criterion I, if $(\mathbf{GBc} \setminus \mathfrak{S}(I)) \neq \emptyset$, then the $\mathcal{S}_0$ of Theorem 5.5 above would witness that separation. Consider $I = \mathbf{TdBc}$. Clearly, $\mathcal{S}_{\text{FinSup}}$ separates **GBc** from **TdBc**. However, the epitomizer $\mathcal{S}_0$ of Theorem 5.5 clearly *is* **TdBc**-learnable—disproving the present first author's old intuition. Nicely, though, from $\mathfrak{O}^{\mathbf{LL}}_{\text{elemWise}}$-robustness of **TdBc** (Example 5.1), the epitomizer $\mathcal{S}_0$ of Theorem 5.7 *does* separate **GBc** from **TdBc**.

References

1 D. Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45:117–135, 1980.

2 J. Bārzdiņš. Inductive inference of automata, functions and programs. In *Proceedings of the 20th International Congress of Mathematicians, Vancouver, Canada*, pages 455–560, 1974. English translation in, *AMS Translations*: Series 2 109 (1977), pp. 107-112.

3 J. Bārzdiņš. Two theorems on the limiting synthesis of functions. *In Theory of Algorithms and Programs, Latvian State University, Riga*, 210:82–88, 1974.

4 L. Blum and M. Blum. Toward a mathematical theory of inductive inference. *Information and Control*, 28:125–155, 1975.

5 J. Case. Periodicity in generations of automata. *Mathematical Systems Theory*, 8:15–32, 1974.

6 J. Case. Infinitary self-reference in learning theory. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:3–16, 1994.

7 J. Case and T. Kötzing. Dynamic modeling in inductive inference. In *Proceedings of ALT*, pages 404–418, 2008.

8 J. Case and T. Kötzing. Dynamically delayed postdictive completeness and consistency in learning. In *Proceedings of ALT*, pages 389–403. Springer, 2008.

9 J. Case and T. Kötzing. Solutions to open questions for non-U-shaped learning with memory limitations. In *Proceedings of ALT*, pages 285–299. Springer, 2010.

10 J. Case and T. Kötzing. Strongly non-U-shaped learning results by general techniques. In *Proceedings of COLT*, 2010. http://www.colt2010.org/papers/011koetzing.pdf.

11 J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.

12 R. Freivalds, E. Kinber, and C. Smith. On the intrinsic complexity of learning. *Information and Computation*, 123(1):64–71, 1995.

13 R. Freivalds, E. Kinber, and R. Wiehagen. Connections between identifying functionals, standardizing operations, and computable numberings. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 30:145–164, 1984.

14 M. Fulk. *A Study of Inductive Inference Machines.* PhD thesis, SUNY at Buffalo, 1985.

15 M. Fulk. Robust separations in inductive inference. In *Proceedings of FOCS*, pages 405–410, St. Louis, Missouri 1990.

16 E. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

17 S. Jain. The intrinsic complexity of learning: A survey. *Fundamenta Informaticae*, 57(1):17–37, 2003.

18 S. Jain, E. Kinber, C. Papazian, C. Smith, and R. Wiehagen. On the intrinsic complexity of learning recursive functions. *Information and Computation*, 184(1):45 – 70, 2003.

19 S. Jain, D. Osherson, J. Royer, and A. Sharma. *Systems that Learn: An Introduction to Learning Theory.* MIT Press, Cambridge, Mass., second edition, 1999.

20 T. Kötzing. *Abstraction and Complexity in Computational Learning in the Limit.* PhD thesis, University of Delaware, 2009. http://pqdtopen.proquest.com/#viewpdf?dispub=3373055.

21 M. Pour-El. A comparison of five "computable" operators. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 6:325–340, 1960.

22 H. Rogers. *Theory of Recursive Functions and Effective Computability.* McGraw Hill, New York, 1967. Reprinted by MIT Press, Cambridge, Massachusetts, 1987.

23 J. Royer and J. Case. *Subrecursive Programming Systems: Complexity and Succinctness.* Research monogr. in *Progress in Theoretical Computer Science.* Birkhäuser Boston, 1994.

24 R. Wiehagen. Limes-Erkennung rekursiver Funktionen durch spezielle Strategien. *Elektronische Informationverarbeitung und Kybernetik*, 12:93–99, 1976.

On Parsimonious Explanations For 2-D Tree- and Linearly-Ordered Data

Howard Karloff[1], Flip Korn[2], Konstantin Makarychev[3], and Yuval Rabani[4]

1 **AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932. E-mail:** `howard@research.att.com`.

2 **AT&T Labs – Research, 180 Park Avenue, Florham Park, NJ 07932. E-mail:** `flip@research.att.com`.

3 **IBM Research, Box 218, Yorktown Heights, NY 10598. E-mail:** `konstantin@us.ibm.com`.

4 **The Rachel and Selim Benin School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem 91904, Israel. E-mail:** `yrabani@cs.huji.ac.il`.

Abstract

This paper studies the "explanation problem" for tree- and linearly-ordered array data, a problem motivated by database applications and recently solved for the one-dimensional tree-ordered case. In this paper, one is given a matrix $A = (a_{ij})$ whose rows and columns have semantics: special subsets of the rows and special subsets of the columns are meaningful, others are not. A submatrix in A is said to be meaningful if and only if it is the cross product of a meaningful row subset and a meaningful column subset, in which case we call it an "allowed rectangle." The goal is to "explain" A as a sparse sum of weighted allowed rectangles. Specifically, we wish to find as few weighted allowed rectangles as possible such that, for all i, j, a_{ij} equals the sum of the weights of all rectangles which include cell (i, j).

In this paper we consider the natural cases in which the matrix dimensions are tree-ordered or linearly-ordered. In the tree-ordered case, we are given a rooted tree T_1 whose leaves are the rows of A and another, T_2, whose leaves are the columns. Nodes of the trees correspond in an obvious way to the sets of their leaf descendants. In the linearly-ordered case, a set of rows or columns is meaningful if and only if it is contiguous.

For tree-ordered data, we prove the explanation problem NP-Hard and give a randomized 2-approximation algorithm for it. For linearly-ordered data, we prove the explanation problem NP-Hard and give a 2.56-approximation algorithm. To our knowledge, these are the first results for the problem of sparsely and exactly representing matrices by weighted rectangles.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.332

1 Introduction

This paper studies two related problems of "explaining" data parsimoniously. In the first part of this paper, we focus on providing a top-down "hierarchical explanation" of "tree-ordered" matrix data. We motivate the problem as follows. Suppose that one is given a matrix $A = (a_{ij})$ of data, and that the rows naturally correspond to the leaves of a rooted tree T_1, and the columns, to the leaves of a rooted tree T_2. For example, T_1 and T_2 could represent hierarchical IP addresses spaces with nodes corresponding to IP prefixes. Each node of either T_1 or T_2 is then said to correspond to the set of rows (or columns, respectively) corresponding to its leaf descendants. Say `128.*` (i.e., the set of all 2^{24} IP addresses beginning with "128", which happens to correspond to the `.edu` domain) is a node in T_1 and `209.85.225.*` (i.e., the set of all 2^8 IP addresses beginning with `209.85.225`, which is `www.google.com`'s

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 332–343
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

domain) is a node in T_2. Then `(128.*, 209.85.225.*)` could, say, represent the amount of traffic flowing from all hosts in the `.edu` domain (e.g., `128.8.127.3`) to all hosts in the `www.google.com` domain (e.g., `209.85.225.99`). It is easy to relabel the rows or columns so that each internal node of T_1 or T_2 corresponds to a contiguous block of rows or columns.

We need a few definitions. Let us say a *rectangle* in an $m \times n$ matrix A is a set $Rect(i_1, i_2, j_1, j_2) = \{i : i_1 \leq i \leq i_2\} \times \{j : j_1 \leq j \leq j_2\}$, for some $1 \leq i_1 \leq i_2 \leq m$, $1 \leq j_1 \leq j_2 \leq n$. Certain rectangles are *allowed*; others are not. Let $\mathcal{R}$ denote the set of allowed rectangles. Say a set of $w(R)$-weighted rectangles R *represents* $A = (a_{ij})$ if for any cell (i, j), the sum of $w(R)$ over cells that contain (i, j) is a_{ij}.

Returning to the Internet example, a pair (u, v), u a node of T_1, v a node of T_2, corresponds to a rectangle. Say that a rectangle is allowed, relative to T_1 and T_2, if it is the cross product of the set of rows corresponding to some node u in T_1 and the set of columns corresponding to some node v in T_2. In this scenario, we attempt to "explain" or "describe" the matrix by writing it as a sum of weighted allowed rectangles. Formally, we wish to assign a weight w_R to each allowed rectangle R such that the set of weighted rectangles represents A.

Of course there is always a solution: one can just assign weights to the 1×1 rectangles. But this is a trivial description of the matrix. Usually more concise explanations are preferable. For this reason we seek an "explanation" with as few nonzero terms as possible. More precisely, we seek to assign a weight w_R to each allowed rectangle R such that the set of weighted rectangles represents A, and such that the number of nonzero weights w_R assigned is minimized. (We define problems formally in Section 3.)

Here is a 1-dimensional example. Suppose that a media retailer sells items in exactly four categories: action-movie DVD's, comedy DVD's, books, and CD's. The retailer builds a hierarchy with four leaves, one for each of the categories of items. A node "DVD's" is the parent of leaves "action-movie DVD's" and "comedy DVD's". There is one more node, a root labeled "all", with children "DVD's", "books", and "CD's".

Suppose that one year, sales of action-movie DVD's increased by \$6000 and sales of the other three categories increased by \$8000 each. One could represent the sales data by giving those four numbers, one for each leaf of the hierarchy, yet one could more parsimoniously say that there was a general increase of \$8000 for all (leaf) categories, in addition to which there was a decrease of \$2000 for action-movie DVD's. This is represented by assigning \$8000 to node "all" and \$-2000 to "action-movie DVD's". While many different linear combinations may be possible, simple explanations tend to be most informative. Therefore, we seek an answer minimizing the explanation size (the number of nonzero terms required in the explanation).

Here is a definition of TREE×TREE. An instance consists of an $m \times n$ matrix $A = (a_{ij})$, along with two rooted trees, a tree T_1 whose leaf set is the set of rows of the matrix, and a tree T_2 whose leaf set is the set of columns. Let $L_i(v)$ be the leaf descendants of node v in tree T_i, $i \in \{1, 2\}$. Now $\mathcal{R}$ is just the set $\{L_1(u) \times L_2(v) \ : \ u$ is a node in T_1 and v is a node in $T_2\}$. The goal is to find the smallest set of weighted rectangles which represents A. We prove this problem NP-hard and give a randomized 2-approximation algorithm for it. APX-hardness is not known.

The second problem, ALLRECTS, is motivated by the need to concisely describe or explain linearly-ordered data. Imagine that one has two ordered parameters, such as horizontal and vertical location, or age and salary. No trees are involved now. Instead we allow any interval of rows (i.e., $\{i : i_1 \leq i \leq i_2\}$ for any $1 \leq i_1 \leq i_2 \leq m$) and any interval of columns (i.e., $\{j : j_1 \leq j \leq j_2\}$ for any $1 \leq j_1 \leq j_2 \leq n$). For example, $[800, 1000] \times [500, 1500]$ could be used to represent a geographical region extending eastward from 800 to 1000 miles and northward from 500 to 1500 miles, and $[35.0, 45.0] \times [80000, 95000]$ could be used to represent the subset of people 35-44 years old and earning a salary of \$80000-\$95000. Then we can use the former "rectangles" to summarize the change (say, in population counts) with respect to location, or use the latter with respect to demographic attributes age and salary.

Hence in AllRects the set $\mathcal{R}$ of allowed rectangles is the cross product between the set of row intervals and the set of column intervals. As a linear combination of how few arbitrary rectangles can we write the given matrix? We prove this problem NP-hard and give a 2.56-approximation algorithm for it. Again, APX-hardness is unknown.

2 Related Work

To our knowledge, while numerous papers have studied similar problems, none proposes any algorithm for either of the two problems we study. One very relevant prior piece of work is a polynomial-time exact algorithm solving the 1-dimensional version of Tree×Tree (more properly called the "tree" case in 1-d, since only one tree is involved) [1]. Here, as in the media-retailer example above, we have a sequence of integers and a tree whose leaves are the elements of the sequence. Indeed, we use this algorithm heavily in constructing our randomized constant-factor approximation algorithm for the tree×tree case.

Relevant to our work is [4] by Bansal, Coppersmith, and Schieber, which (in our language) studies the 1-d (exact) problem in which all intervals are allowed and all must have *nonnegative* weights, proves the problem NP-hard, and gives a constant-factor approximation algorithm.

Also very relevant is a paper by Natarajan [13], which studies an "inexact" version of the problem: instead of finding weighted rectangles whose sum of weights is a_{ij} exactly, for each matrix cell (i, j), these sums approximate the a_{ij}'s. (Natarajan's algorithm is more general and can handle any arbitrary set $\mathcal{R}$ of allowed rectangles; however, the algorithm is very slow.) More precisely, in the output set of rectangles, define a'_{ij} to be the sum of the weights of the rectangles containing cell (i, j). Natarajan's algorithm ensures, given a tolerance $\Delta > 0$, that the L_2 error $\sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} (a'_{ij} - a_{ij})^2}$ is at most Δ. (Natarajan's algorithm cannot be used for $\Delta = 0$.) The upper bound on the number of rectangles produced by Natarajan's algorithm is a factor of approximately $18 \ln(||A||_2/\Delta)$ (where $||A||_2$ is the square root of the sum of squares of the entries of A) larger than the optimal number used by an adversary who is allowed, instead, only L_2-error $\Delta/2$. Furthermore, Natarajan's algorithm is very slow, much slower than our algorithms. See the full version of our paper for details.

Frieze and Kannan in [9] show how to inexactly represent a matrix as a sum of a small number of rank-1 matrices, but their method is unsuitable to solve our problem, as not only is there no way to restrict the rank-1 matrices to be rectangles, the error is of L_1 type rather than L_∞. In other words, the *sum* of the mn errors is bounded by Δmn, rather than individual errors' being bounded by Δ.

Our problem may remind readers of compressed sensing, the decoding aspect of which requires one to seek a solution x with fewest nonzeroes to a linear system $Hx = b$. The key insight of compressed sensing is that when H satisfies the "restricted isometry property" [16, 6, 8], as do almost all random matrices, the solution x of minimum L_1 norm is also the sparsest solution. The problem with applying compressed sensing to the problems mentioned herein, when the matrix A is $m \times n$, is that the associated matrix H, which has mn rows and a number of columns equal to the number of allowed rectangles, is anything but random. On a small set of test instances, the authors found the solutions of minimum L_1 norm (using linear programming) and discovered that they were far from sparsest.

Other authors have studied other ways of representing matrices. Applegate et al. [2] studied the problem of representing a *binary* matrix, starting from an all-zero matrix, by an *ordered* sequence of rectangles, each of whose entries is all 0 or all 1, in which a_{ij} should equal the entry of the *last* rectangle which contains cell (i, j). Anil Kumar and Ramesh [3] study the same model in which only all-1 rectangles are allowed (in which case the order clearly doesn't matter). Two papers [14, 11] study the Gale-Berlekamp switching game and can be thought of as a variant of our problem over $\mathbb{Z}_2$.

3 Formal Definitions and Examples

Given an $m \times n$ matrix $A = (a_{ij})$ and $1 \le i_1 \le i_2 \le m$, $1 \le j_1 \le j_2 \le n$, recall that $Rect(i_1, i_2, j_1, j_2) = \{(i, j) | i_1 \le i \le i_2, j_1 \le j \le j_2\}$. Define $Rects = \{Rect(i_1, i_2, j_1, j_2) | 1 \le i_1 \le i_2 \le m, 1 \le j_1 \le j_2 \le n\}$. For each of the two problems, we are given a subset $\mathcal{R} \subseteq Rects$; the only difference between the two problems we discuss is the definition of $\mathcal{R}$. The goal is to find a smallest subset $OPT_2(A)$ of $\mathcal{R}$, and an associated weight $w(R)$ (positive or negative) for each rectangle R, such that every cell (i, j) is covered by rectangles whose weights sum to a_{ij}, that is,

$$a_{ij} = \sum_{R: R \in OPT_2(A) \text{ and } R \ni (i,j)} w(R), \tag{1}$$

the "2" in "$OPT_2(A)$" referring to the fact that A is 2-dimensional.

While the algorithm for the tree×tree case appears (in Section 4) before that for the arbitrary-rectangles case (in Section 5), here we define AllRects, the latter, first, since it's easier to define. As mentioned above, we call the case of $\mathcal{R} = Rects$ AllRects.

Example. Since the matrix

$$A = \begin{bmatrix} 2&2&2&2 \\ 5&3&1&2 \\ 6&4&1&3 \\ 5&5&2&2 \end{bmatrix} = 2\begin{bmatrix} 1&1&1&1 \\ 1&1&1&1 \\ 1&1&1&1 \\ 1&1&1&1 \end{bmatrix} + 3\begin{bmatrix} 0&0&0&0 \\ 1&1&0&0 \\ 1&1&0&0 \\ 1&1&0&0 \end{bmatrix} + 1\begin{bmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 1&1&1&1 \\ 0&0&0&0 \end{bmatrix} - 2\begin{bmatrix} 0&0&0&0 \\ 0&1&1&0 \\ 0&1&1&0 \\ 0&0&0&0 \end{bmatrix} + 1\begin{bmatrix} 0&0&0&0 \\ 0&0&1&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix},$$

A can be written as a linear combination with $w(\{1, 2, 3, 4\} \times \{1, 2, 3, 4\}) = 2$, $w(\{2, 3, 4\} \times \{1, 2\}) = 3$, $w(\{3\} \times \{1, 2, 3, 4\}) = 1$, $w(\{2, 3\} \times \{2, 3\}) = -2$, and $w(\{2\} \times \{3\}) = 1$. Hence $|OPT_2(A)| \le 5$.

We need some notation in order to define Tree×Tree, in which we are also given trees T_1 and T_2. We use R_i to denote the row vector in the ith row of the input matrix, $1 \le i \le m$. For a node $u \in T_1$, let $S^1_u = \{R_l \ : \ l \text{ is a leaf descendant in } T_1 \text{ of } u\}$. Similarly, we use C_j to denote the column vector in the jth column of the input matrix, $1 \le j \le n$. For a node $v \in T_2$, let $S^2_v = \{C_l \ : \ l \text{ is a leaf descendant in } T_2 \text{ of } v\}$. Note that, since T_1 and T_2 are trees, $\{S^1_u | u \in T_1\}$ and $\{S^2_v | v \in T_2\}$ are laminar.

In this notation, in Tree×Tree, $\mathcal{R} = \{S^1_u | u \in T_1\} \times \{S^2_v | v \in T_2\}$.

Example. Using trees T_1, T_2 having a root with four children (and no other nodes) apiece, we may use any single row or all rows, and any single column or all columns. For example, since the matrix

$$\begin{aligned} A = \begin{bmatrix} 5&3&4&5 \\ 3&0&2&4 \\ 2&2&1&3 \\ 3&3&2&3 \end{bmatrix} &= 3\begin{bmatrix} 1&1&1&1 \\ 1&1&1&1 \\ 1&1&1&1 \\ 1&1&1&1 \end{bmatrix} + 2\begin{bmatrix} 1&1&1&1 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix} - 1\begin{bmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 1&1&1&1 \\ 0&0&0&0 \end{bmatrix} - 1\begin{bmatrix} 0&0&1&0 \\ 0&0&1&0 \\ 0&0&1&0 \\ 0&0&1&0 \end{bmatrix} \\ &- 2\begin{bmatrix} 0&1&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix} - 3\begin{bmatrix} 0&0&0&0 \\ 0&1&0&0 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix} + 1\begin{bmatrix} 0&0&0&0 \\ 0&0&0&1 \\ 0&0&0&0 \\ 0&0&0&0 \end{bmatrix} + 1\begin{bmatrix} 0&0&0&0 \\ 0&0&0&0 \\ 0&0&0&1 \\ 0&0&0&0 \end{bmatrix}, \end{aligned}$$

we can write A as a sum with $w(\{1, 2, 3, 4\} \times \{1, 2, 3, 4\}) = 3$, $w(\{1\} \times \{1, 2, 3, 4\}) = 2$, $w(\{3\} \times \{1, 2, 3, 4\}) = -1$, $w(\{1, 2, 3, 4\} \times \{3\}) = -1$, $w(\{1\} \times \{2\}) = -2$, $w(\{2\} \times \{2\}) = -3$, $w(\{2\} \times \{4\}) = 1$, and $w(\{3\} \times \{4\}) = 1$. Since there are eight matrices, $|OPT_2(A)| \le 8$.

Note that we use the same notation, $OPT_2(A)$, for the optimal solutions of both AllRects and Tree×Tree.

4 Approximation Algorithm for Tree×Tree

We defer the (interesting) proof of NP-Hardness of Tree×Tree to the full version of the paper. Our algorithm will rely upon the exact algorithm, due to Agarwal et al. [1], for the case in which the matrix has just one column (that is, the 1-dimensional case).

▶ **Definition 1.** Given a fixed rooted tree T_1 with m leaves, and an m-vector $V = (v_i)$, let $OPT_1(V)$ denote a smallest set of intervals $I = \{i : i_1 \leq i \leq i_2\} \subseteq [1, m]$ and associated weights $w(I)$, each I corresponding to a node of T_1, such that for all i, $v_i = \sum_{I:I \in OPT_1(V) \text{ and } I \ni i} w(I)$.

Clearly $|OPT_1(V)|$ equals $|OPT_2(V')|$, where V' is the $m \times 1$ matrix containing V as a column. The difference is that $OPT_1(V)$ is a set of vectors while $OPT_2(V')$ is a set of rectangles. We emphasize that V is a vector and that the definition depends on T_1 and not T_2 by putting the "1" in "$OPT_1(V)$". The key point is that [1] showed how to compute $OPT_1(V)$ exactly.

In order to charge the algorithm's cost against $OPT_2(A)$, we need to know some facts about $OPT_2(A)$. Recall that $OPT_2(A)$ is a smallest subset of $\mathcal{R}$ such that there are weights $w(R)$ such that equation (1) holds.

▶ **Definition 2.**

1. For each rectangle R and associated weight w_R, let R'_{w_R} denote the $m \times n$ matrix which is 0 for every cell (i, j), except that $(R'_{w_R})_{ij} := w_R$ if $(i, j) \in R$.
2. Given a vertex v of T_2, let D_v be the set of all $R \in OPT_2(A)$ such that R has column set exactly equal to S^2_v.
3. Now let $K_v = \sum_{R \in D_v} R'_{w_R}$. By definition of D_v, all columns j of K_v for $j \in D_v$ are the same. Let V_v be column j of K_v for any $j \in D_v$.

▶ **Lemma 3.** *The column vectors (V_v) satisfy the following:*

1. *For all leaves l in T_2, the vector C_l equals the sum of V_v over all ancestors v of l in T_2.*
2. *For all leaves l' and l'' in T_2 with a common ancestor u, the vector $C_{l'} - C_{l''}$ equals the sum of V_v over all vertices v on the path from u down to l' (not including $v = u$) minus the sum of V_v over all vertices v on the path from u down to l'' (not including $v = u$).*
3. *The union, over all vertices $v \in T_2$, of $OPT_1(V_v) \times \{S^2_v\}$ (which obviously has size $|OPT_1(V_v)|$), with the corresponding weights, is an optimal solution for* Tree×Tree *on A.*
4. *$|OPT_2(A)| = \sum_{v \in T_2} |OPT_1(V_v)|$.*

Proof. The nodes v which correspond to sets of columns containing column C_l are exactly the ancestors in T_2 of l. Hence, Part 1 follows.

Part 2 is an immediate corollary of Part 1. Clearly, by Part 1, the union over all vertices $v \in T_2$ of $OPT_1(V_v) \times \{S^2_v\}$ is a feasible solution for Tree×Tree on A. It is also optimal, and here is a proof. The size of the optimal solution $OPT_2(A)$ equals the sum, over vertices $v \in T_2$, of the number of rectangles in $OPT_2(A)$ having column set S^2_v. Fix a vertex $v \in T_2$. Since the weighted sum of the rectangles in $OPT_2(A)$ with column set S^2_v is V_v, and each has a row set S^1_u for some $u \in T_1$, the number of such rectangles must be at least $OPT_1(V_v)$. If the number of rectangles with column set S^2_v strictly exceeded $OPT_1(V_v)$, we could replace all rectangles in $OPT_2(A)$ having column set S^2_v by a smaller set of weighted rectangles having column set S^2_v, each of whose columns is the same, and summing to V_v in each column; since the new set and the old set have the same weighted sum, the new solution would still sum to A, and have better-than-optimal size, thereby contradicting optimality of $OPT_2(A)$. Part 3 follows.

Part 4 follows from Part 3. ◀

Lemma 3 will be instrumental in analyzing the algorithm.

While the algorithm is very simple to state, it was nontrivial to develop and analyze. In the algorithm, we use the algorithm by Agarwal et al. [1] to obtain $OPT_1(V)$ given a vector V.

Algorithm for TREE×TREE

```
1. For every internal node $u$ in the tree $T_2$, pick a random child $u^*$ of $u$ and let $c(u) = u^*$.
   Let $path(u)$ be the random path going from $u$ to a leaf: $u \mapsto c(u) \mapsto c(c(u)) \mapsto \cdots \mapsto l(u)$,
   where we denote the last node on the path, the leaf, by $l(u)$.
2. Where $root$ denotes the root of $T_2$, for every node $u$ in $T_2$, in increasing order by depth, do:
   - If $u$ is the root of $T_2$, then
      - Output $OPT_1(C_{l(root)}) \times \{S^2_{root}\}$ with the corresponding weights (those of the optimal solution for $C_{l(root)}$).
   - Else
      - Let $p(u)$ be the parent of $u$.
      - Output $OPT_1(C_{l(u)} - C_{l(p(u))}) \times \{S^2_u\}$ with the corresponding weights.
```

▶ **Theorem 4.** *The expected cost of the algorithm is at most* $2|OPT_2(A)|$.

In the main part of the paper we prove a weaker guarantee for exposition: the expected cost of the algorithm is at most $4|OPT_2(A)|$. We defer the improvement to the full version of the paper. The algorithm can be easily derandomized using dynamic programming.

Proof. Every column C_u is covered by rectangles with sum

$$(C_u - C_{l(p(u))}) + (C_{l(p(u))} - C_{l(p(p(u)))}) + \cdots + C_{l(root)} = C_u.$$

Thus the algorithm produces a valid solution. We now must estimate the expected cost of the solution. The total cost incurred by the algorithm is

$$|OPT_1(C_{l(root)})| + \sum_{u \neq root} |OPT_1(C_{l(u)} - C_{l(p(u))})|.$$

Assume, without loss of generality, that all nodes in the tree either have two or more children or are leaves. Denote the number of children of a node v, the degree of v, by $d(v)$. Denote by $\mathbf{1}$ the indicator function. Observe that for the root node we have

$$|OPT_1(C_{l(root)})| = \left| OPT_1\Big(\sum_{v \in path(root)} V_v \Big) \right| \leq \sum_{v \in path(root)} |OPT_1(V_v)|;$$

for a nonroot vertex u, we have by Lemma 3 (2), keeping in mind that $l(\cdot)$, $c(\cdot)$, and $path(\cdot)$ are random,

$$\begin{aligned} |OPT_1(C_{l(u)} - C_{l(p(u))})| &= \left| OPT_1\Big(\sum_{v \in path(u)} V_v - \sum_{v \in path(c(p(u)))} V_v \Big) \right| \\ &\leq \Big(\sum_{v \in path(u)} |OPT_1(V_v)| + \sum_{v \in path(c(p(u)))} |OPT_1(V_v)| \Big) \cdot \mathbf{1}(u \neq c(p(u))). \end{aligned}$$

Here we used the triangle inequality for the function $|OPT_1(\cdot)|$.

Consider the second sum in the right-hand side. For every child u' of $p(u)$, the random node $c(p(u))$ takes value u' with probability $1/d(p(u))$. Thus

$$\begin{aligned}&\mathbb{E}\Big[\sum_{v\in path(c(p(u)))} |OPT_1(V_v)|\cdot \mathbf{1}(u\neq c(p(u)))\Big]\\ &\quad= \frac{1}{d(p(u))}\sum_{u':u' \text{ is a sibling of } u}\mathbb{E}\Big[\Big(\sum_{v\in path(c(p(u)))}|OPT_1(V_v)|\Big)\ \mid c(p(u))=u'\Big]\\ &\quad= \frac{1}{d(p(u))}\sum_{u':u' \text{ is a sibling of } u}\mathbb{E}\Big[\sum_{v\in path(u')}|OPT_1(V_v)|\Big].\end{aligned}$$

$\Pr\Big(u\neq c(p(u))\Big)$ equals $(d(p(u))-1)/d(p(u))$. Denote this expression by α_u. The total expected size of the solution returned by the algorithm is bounded by

$$\mathbb{E}\Big[\sum_{v\in path(root)}|OPT_1(V_v)|\Big]+\sum_{u\neq root}\alpha_u\mathbb{E}\Big[\sum_{v\in path(u)}|OPT_1(V_v)|\Big] \tag{2}$$

$$+\textstyle\sum_{u\neq root}\frac{1}{d(p(u))}\sum_{u':u' \text{ is a sibling of } u}\mathbb{E}\Big[\sum_{v\in path(u')}|OPT_1(V_v)|\Big]$$

$$=\mathbb{E}\Big[\textstyle\sum_{v\in path(root)}|OPT_1(V_v)|\Big]+\sum_{u\neq root}\alpha_u\mathbb{E}\Big[\sum_{v\in path(u)}|OPT_1(V_v)|\Big]$$

$$+\textstyle\sum_{u'\neq root}\Big(\sum_{u\neq root}\frac{\mathbf{1}(u' \text{ is a sibling of } u)}{d(p(u'))}\Big)\mathbb{E}\Big[\sum_{v\in path(u')}|OPT_1(V_v)|\Big]. \tag{3}$$

Notice that, for a fixed $u'\neq root$,

$$\sum_{u\neq root}\frac{\mathbf{1}(u' \text{ is a sibling of } u)}{d(p(u'))}=\frac{d(p(u'))-1}{d(p(u'))}=\alpha_{u'}<1. \tag{4}$$

Hence, the total cost of the solution is bounded by

$$\sum_u\mathbb{E}\Big[\sum_{v\in path(u)}|OPT_1(V_v)|\Big]+\sum_{u'\neq root}\mathbb{E}\Big[\sum_{v\in path(u')}|OPT_1(V_v)|\Big]\leq 2\sum_u\mathbb{E}\Big[\sum_{v\in path(u)}|OPT_1(V_v)|\Big].$$

Finally, observe that node v belongs to $path(v)$ with probability 1; it belongs to the $path(p(v))$ with probability at most $1/2$; it belongs to the path $path(p(p(v)))$ with probability at most $1/4$, etc. It belongs to $path(u)$ with probability 0 if u is not an ancestor of v. Thus

$$\begin{aligned}2\sum_u\mathbb{E}\Big[\sum_{v\in path(u)}|OPT_1(V_v)|\Big] &= 2\sum_v|OPT_1(V_v)|\cdot\Big(\sum_u\Pr\Big(v\in path(u)\Big)\Big)\\ &\leq 2\sum_v|OPT_1(V_v)|\cdot\Big(1+1/2+1/4+\cdots\Big)\\ &< 4\sum_v|OPT_1(V_v)|\leq 4|OPT_2(A)|.\end{aligned}$$

We have proven that the algorithm finds a 4-approximation. A slightly more careful analysis, in the full version of the paper, shows that the approximation ratio of the algorithm is at most 2. ◀

What is the running time of the 2-approximation algorithm? The time needed to run the 1-dimensional algorithm of [1] is $O(dn)$ where there are n leaves in each tree and the *smaller* of the two depths is d. One can verify that the running time of our 2-approximation algorithm is a factor $O(n)$ larger, or $O(dn^2)$. In most applications at least one of the trees would have depth $O(\log n)$, giving $O(n^2\log n)$ in total.

5 Approximation Algorithm For AllRects

5.1 The 1-Dimensional Problem

First we consider the one-dimensional case, for which we will give a $(23/18+\varepsilon)$-approximation algorithm; $23/18 < 1.278$. We are given a sequence $a_1, a_2, \ldots, a_n$ of numbers and we need to find a collection of closed intervals $[i,j]$ with arbitrary real weights w_{ij} so that every integral point $k \in \{1,\ldots,n\}$ is covered by a set of intervals with total weight a_k. That is, for all k,

$$\sum_{i,j:k\in[i,j]} w_{ij} = a_k. \tag{5}$$

Our goal is to find the smallest possible collection. We shall use the approach of Bansal, Coppersmith, and Schieber [4] (in their problem all $a_i \geq 0$ and all $w_{ij} > 0$). Set $a_0 = 0$ and $a_{n+1} = 0$. Observe that if $a_k = a_{k+1}$, then in the optimal solution every interval covering k also covers $k+1$. On the other hand, since every rectangle covering both k and $k+1$ contributes the same weight to a_k and a_{k+1}, if $a_k \neq a_{k+1}$, then there should be at least one interval that either covers k but not $k+1$, or covers $k+1$ but not k. By the same reason, the difference $a_{k+1} - a_k$, which we denote by $\Delta_k = a_{k+1} - a_k$, equals the difference between the weight of intervals with the left end-point at $k+1$ and the weight of rectangles with the right endpoint at k:

$$\Delta_k = \sum_{j:k+1\leq j} w_{k+1,j} - \sum_{i:i\leq k} w_{ik}. \tag{6}$$

Note that if we find a collection of rectangles with weights satisfying (6), then this collection of intervals is a valid solution to our problem, i.e., then equality (5) holds. Define a directed graph on vertices $\{0,\ldots,n\}$. For every interval $[i,j]$, we add an arc going from $i-1$ to j. Then the condition (6) can be restated as follows: The sum of weights of arcs outgoing from k minus the sum of weights of arcs entering k equals Δ_k. Our goal is to find the smallest set of arcs with non-zero weights satisfying this property. Consider an arbitrary solution and one of the weakly connected components S. The sum $\sum_{k\in S} \Delta_k = 0$, since every arc is counted twice in the sum, once with the plus sign and once with the minus sign. Since S is a connected component the number of arcs connecting nodes in S is at least $|S|-1$. Thus a lower bound on the number of arcs or intervals in the optimal solution is the minimum of

$$\sum_{t=1}^{M} (|S_t| - 1) = n + 1 - M$$

among all partitions of the set of items $\{0,\ldots,n\}$ into M disjoint sets $S_1,\ldots,S_M$ such that $\sum_{k\in S_t} \Delta_k = 0$ for all t. On the other hand, given such a partition $(S_1,\ldots,S_M)$, we can easily construct a set of intervals. Let k_t be the minimal element in S_t. For every element k in $S_t \setminus \{k_t\}$, we add an interval $[k_t+1, k]$ with weight $-\Delta_k$. We now verify that these intervals satisfy (6). If k belongs to S_t and $k \neq k_t$, then there is only one interval in the solution with right endpoint at k. This interval is $[k_t+1, k]$ and its weight is $-\Delta_k$. The solution does not contain intervals with left endpoint at $k+1$ (since $k \neq k_t$). Thus (6) holds as well. If k belongs to S_t and $k = k_t$, the solution does not contain intervals with the right endpoint at k, but for all $k' \in S_t$ there is an interval $[k+1, k']$ with weight $-\Delta_{k'}$. The total weight of these intervals equals

$$\sum_{k'\in S_t;k'\neq k} -\Delta_{k'} = -\sum_{k'\in S_t} \Delta_{k'} + \Delta_k = \Delta_k.$$

Condition (6) again holds.

Thus the problem is equivalent to the problem of partitioning the set of items $\{0,\dots,n\}$ into a family of M sets $\{S_1,\dots,S_M\}$ satisfying the condition $\sum_{k\in S_t}\Delta_k = 0$ for all t, so as to minimize $\sum_t(|S_t|-1) = (n+1)-M$. Notice that the sum of all Δ_k equals 0. Moreover, every set with the sum of Δ_k equal to 0 corresponds to an instance of the 1-dimensional rectangle covering problem. We shall refer to the problem as Zero-Weight Partition.

We now describe the approximation algorithm for Zero-Weight Partition which is a modification of the algorithm of Bansal, Coppersmith, and Schieber [4] designed for a slightly different problem (that of minimizing setup times in radiation therapy).

▶ Remark. For Zero-Weight Partition, our algorithm gives a slightly better approximation guarantee than that of [4]: $23/18 \approx 1.278$ vs $9/7 \approx 1.286$. The difference between algorithms is that the algorithm of Bansal, Coppersmith, and Schieber [4] performs either the first and third steps (in terms of our algorithm; see below), or the second and third steps; while our algorithm always performs all three steps.

In the first step the algorithm picks all singleton sets $\{k\}$ with $\Delta_k = 0$ and pairs $\{i,j\}$ with $\Delta_i = -\Delta_j$. It removes the items covered by any of the chosen sets. At the second step, with probability 2/3 the algorithm enumerates all triples $\{i,j,k\}$ with $\Delta_i+\Delta_j+\Delta_k = 0$ and finds the largest 3-set packing among them using the $(3/2+\varepsilon)$-approximation algorithm due to Hurkens and Schrijver [10], i.e., it finds the largest (up to a factor of $(3/2+\varepsilon)$) disjoint family of triples $\{i,j,k\}$ with $\Delta_i+\Delta_j+\Delta_k = 0$. Otherwise (with probability 1/3), the algorithm enumerates all quadruples $\{i,j,k,l\}$ having $\Delta_i+\Delta_j+\Delta_k+\Delta_l = 0$ and finds the largest 4-set packing among them using the $(2+\varepsilon)$-approximation algorithm due to Hurkens and Schrijver [10]. At the third, final, step the algorithm covers all remaining items, whose sum of Δ_k's is zero, with one set.

Before we start analyzing the algorithm, let us consider a simple example. Suppose that $(a_1,a_2,a_2,a_4,a_5,a_6) = (15,8,10,17,18,15)$. First we surround the vector with two 0's: $(a_0,a_1,a_2,a_2,a_4,a_5,a_6,a_7) = (0,15,8,10,17,18,15,0)$. Then compute the vector of Δ_k's: $(\Delta_0,\Delta_1,\Delta_2,\Delta_2,\Delta_4,\Delta_5,\Delta_6) = (15-0,8-15,10-8,17-10,18-17,15-18,0-15) = (15,-7,2,7,1,-3,-15)$. Notice that $(-15)+7+(-2)+(-7)+(-1)+3+15 = 0$. We partition the set into sets of weight 0: $\{\Delta_0,\Delta_6\},\{\Delta_1,\Delta_3\},\{\Delta_2,\Delta_4,\Delta_5\}$. This partition corresponds to the following solution of the 1-dimensional problem: interval $[1,6]$ with weight 15, interval $[2,3]$ with weight -7, interval $[3,4]$ with weight -1, interval $[3,5]$ with weight 3.

▶ **Lemma 5.** *For every positive $\varepsilon > 0$, the approximation ratio of the algorithm when using ε is at most $23/18 + O(\varepsilon)$, with $23/18 < 1.278$.*

Proof. First, observe that the partitioning returned by the algorithm is a valid partitioning, i.e., every item belongs to exactly one set and the sum of Δ_k's in every set equals 0. We show that the first step of the algorithm is optimal. That is, there exists an optimal solution that contains exactly the same set of singletons and pairs as in the partition returned by the algorithm. Suppose that the optimal solution breaks one pair $\{i,j\}$ ($\Delta_i = -\Delta_j$) and puts i in S and j in T. Then we can replace sets S and T with two new sets $\{i,j\}$ and $S\cup T\setminus\{i,j\}$. The new solution has the same cost as before; the sum of Δ_k's in every set is 0, but the pair $\{i,j\}$ belongs to the partitioning. Repeating this procedure several times, we can transform an arbitrary optimal solution into an optimal solution that contains the same set of singletons and pairs as the solution obtained by the approximation algorithm.

For the sake of the presentation let us assume that $\varepsilon = 0$ (that is, we assume that the approximation algorithms due to Hurkens and Schrijver [10], we use in our algorithm, have approximation guarantees at most 3/2 and 2). Let p_k be the number of sets of size k in the optimal solution. The cost of the optimal solution is $p_2 + 2p_3 + 3p_4 + 4p_5 + \cdots$, because the objective function charges $|S| - 1$ to a set of size $|S|$. Our approximation algorithm also finds p_1 singleton sets and p_2 pairs. Then with probability $2/3$, it finds $s_3 \geq (2/3)p_3$ triples and covers the remaining $3 \cdot (p_3 - s_3) + 4p_4 + 5p_5 + \cdots$ vertices with one set; and with probability $1/3$, it finds $s_4 \geq p_4/2$ quadruples and covers the remaining $3p_3 + 4 \cdot (p_4 - s_4) + 4p_4 + 5p_5 + \cdots$ vertices with one set. Thus the expected cost of the solution returned by the algorithm equals

$$\frac{2}{3}\Big(p_2 + 2 \cdot \frac{2p_3}{3} + 3 \cdot \frac{p_3}{3} + 4p_4 + \sum_{k \geq 5} kp_k - 1\Big) + \frac{1}{3}\Big(p_2 + 3 \cdot \frac{p_4}{2} + 3p_3 + 4 \cdot \frac{p_4}{2} + \sum_{k \geq 5} kp_k - 1\Big)$$
$$= p_2 + \frac{23}{9}p_3 + \frac{23}{6}p_4 + \sum_{k \geq 5} kp_k - 1. \quad (7)$$

Therefore, the approximation ratio of the algorithm, assuming that $\varepsilon = 0$, is

$$\frac{p_2 + \frac{23}{9}p_3 + \frac{23}{6}p_4 + \sum_{k \geq 5} kp_k - 1}{p_2 + 2p_3 + 3p_4 + \sum_{k \geq 5}(k-1)p_k} \leq \max\left\{\frac{1}{1}, \frac{\frac{23}{9}}{2}, \frac{\frac{23}{6}}{3}, \frac{5}{4}, \frac{6}{5}, \dots\right\} = \frac{23}{18}.$$

It is easy to verify that if $\varepsilon > 0$, the approximation ratio of the algorithm is at most $23/18 + O(\varepsilon)$. ◀

In the full version of the paper we prove that finding the exact solution of the problem is NP-hard.

5.2 The 2-Dimensional Case

We now consider the 2-dimensional case (which does not appear in [4]). We are given an $m \times n$ matrix $A = (a_{ij})$ $(1 \leq i \leq m,\ 1 \leq j \leq n)$ and we need to cover it with the minimum number of weighted rectangles $Rect(i_1, i_2, j_1, j_2)$ (for arbitrary i_1, i_2, j_1, j_2); we use $w(i_1, i_2, j_1, j_2)$ for the weight of $Rect(i_1, i_2, j_1, j_2)$. We assume that $a_{ij} = 0$ for i and j outside the rectangle $\{1, \dots, m\} \times \{1, \dots, n\}$.

By analogy to the 1-dimensional case, define $\Delta_{ij} = a_{i,j} - a_{i,j+1} + a_{i+1,j+1} - a_{i+1,j}$. Call a pair (i,j) with $0 \leq i \leq m$, $0 \leq j \leq n$, with $\Delta_{ij} \neq 0$ an *array corner*. Imagine that the matrix is written in an $m \times n$ table, and Δ_{ij}'s are written at the grid nodes. The key point is that every rectangle covers exactly one, two, or four of the cells $(i+1, j+1)$, (i,j), $(i, j+1)$, $(i+1, j)$ bordering a grid point, and that those covering two or four of those cells cannot affect Δ_{ij}. This means that only rectangles having a corner at the intersection of the ith and jth grid line contribute to Δ_{ij}. (This is why the definition of Δ_{ij} was "by analogy" to the 1-d case.) This means that the number of rectangles in the optimal solution must be at least one quarter of the number of array corners, the "one-quarter" arising from the fact that each rectangle has exactly four corners and can hence be responsible for at most four of the array corners.

It is easy now to give a 4-approximation algorithm, which we sketch without proof, based on this observation. Build a matrix M, initially all zero, which will eventually equal the input matrix A. Until no more array corners exist in $A - M$, find an array corner (i,j) with $i < m$ and $j < n$. (As long as array corners exist, there must be one with $i < m$ and $j < n$.) Let $\Delta \neq 0$ be Δ_{ij}. Add to M a rectangle of weight Δ with upper left corner at (i,j) and extending as far as possible to the right and downward, eliminating the array corner at (i,j) in $A - M$.

It is easy to see that (1) when the algorithm terminates, $M = A$, and that (2) the number of rectangles used is at most the number of array corners in A, and hence at most $4|OPT_2(A)|$.

Now we give, instead, a more sophisticated, $23/9+\varepsilon < 2.56$-approximation algorithm for the 2D problem. The idea is to make more efficient use of the rectangles. Instead of using only one corner of each (in contrast to the adversary, who might use all four), now we will use two. In fact, we will deal separately with different horizontal (between-consecutive-row) grid lines, using a good 1-dimensional approximation algorithm to decide how to eliminate the array corners on that grid line. Every time the 1-d algorithm tells us to use an interval $[j_1, j_2]$, we will instead inject a rectangle which starts in column j_1 and ends in column j_2, and extends all the way to the bottom. Because we use 2 of each rectangle's 4 corners, we pay a price of a factor of 4/2 over the 1-d approximation ratio of $23/18 + O(\varepsilon)$. Hence we will get $23/9 + O(\varepsilon)$.

Here are the details. Fix i and consider the restriction of the zero-weight partition problem to the ith horizontal grid line, i.e., the 1-dimensional zero-weight partition problem with $\Delta_j = \Delta_{ij}$. Denote by OPT^i the cost of the optimal solution. The number of rectangles touching the ith horizontal grid line from above or below is at least OPT^i, since only these rectangles contribute Δ_{ij}'s. Every rectangle touches only two horizontal grid lines, thus the total number of rectangles is at least $\sum_{i=1}^{m} OPT^i/2$.

All rectangles generated by our algorithm will touch the bottom line of the table; that is why we lose a factor of 2. Note that if we could solve the 1-dimensional problem exactly we would be able to find a covering with $\sum_{i=1}^{m} OPT^i$ rectangles and thus get a 2 approximation. For each horizontal grid line i, the algorithm solves the 1-dimensional problem (with $\Delta_j = \Delta_{ij}$) and finds a set of intervals $[j_1, j_2]$ with weights $w_{j_1 j_2}$. These intervals are the top sides of the rectangles generated by the algorithm. All bottom sides of the rectangles lie on the bottom grid line of the table. That is, for every interval $[j_1, j_2]$ the algorithm adds the rectangle $Rect(i, m, j_1, j_2)$ to the solution and sets its weight $w(i, m, j_1, j_2)$ to be $w_{j_1 j_2}$.

The total number of rectangles in the solution output by the algorithm is $\sum_{i=1}^{m} ALG_i$, where ALG_i is the cost of the solution of the 1-dimensional problem. Thus the cost of the solution is at most $2 \cdot (23/18 + O(\varepsilon))$ times the cost of the optimum solution. We now need to verify that the set of rectangles output by the algorithm is indeed is a solution.

Subtract the weight of each rectangle from all a_{ij}'s covered by the rectangle. We need to prove that the residual matrix

$$a'_{ij} = a_{ij} - \sum_{i_1, j_1, j_2 : (i,j) \in Rect(i_1, m, j_1, j_2)} w(i_1, m, j_1, j_2)$$

equals zero. Observe that $\Delta'_{ij} = a'_{i+1,j+1} + a'_{ij} - a'_{i+1,j} - a'_{i,j+1} = 0$ for all $0 \le i \le m-1$ (i.e., all rows i, possibly, except for the bottom line) and $0 \le j \le n$. Assume that not all a'_{ij} equal to 0. Let $a'_{i_0 j_0}$ be the first nonzero a'_{ij} with respect to the lexicographical order on (i, j). Then $a'_{i_0-1, j_0-1} = a'_{i_0-1, j_0} = a'_{i_0, j_0-1} = 0$. Thus $a'_{i_0 j_0} = 0$. We have proven the following theorem.

▶ **Theorem 6.** *For every positive ε, there exists a polynomial-time approximation algorithm for* AllRects *with approximation guarantee at most $23/9 + O(\varepsilon)$, with $23/9 = 2.5555....$.*

5.3 A Simplified Algorithm

Because of the dependence on ε, the running time of the previous algorithm can be large when ε is small. A simpler algorithm for the 1-dimensional case—namely, just use pairs and triples—can be shown to give ratio 4/3 for the 1-d case, and hence $8/3 = 2.6666...$ in 2-d, only slightly worse than 23/9. For the simplified 1-d algorithm, the running time is $O(n + k^2 \log k)$, if there are k Δ's. To run the 2-d algorithm, the running time becomes

$O(n^2+\sum_{i=1}^n k_i^2 \log k_i)$, where there are k_i corners on the ith row. Since the number of corners is $\Theta(OPT)$, the running time is at most $O(n^2)$ plus $O(\max_{k_1+k_2+\cdots+k_n=OPT} \sum_i k_i^2 \log k_i)$. Since $f(x) = x^2 \log x$ is convex, this quantity is maximized by making as many k_i's equal to n as possible. A simple proof then shows that the time is $O(n^2 + OPT \cdot (n \log n))$.

6 Acknowledgment

We thank Divesh Srivastava for initial conversations which inspired this work.

References

1 D. Agarwal, D. Barman, D. Gunopulos, N. Young, F. Korn, and D. Srivastava, “Efficient and Effective Explanation of Change In Hierarchical Summaries," Proc. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2007, 6-15.

2 D. Applegate, G. Calinescu, D. S. Johnson, H. Karloff, K. Ligett, and J. Wang. “Compressing Rectilinear Pictures and Minimizing Access Control Lists," SODA 2007, 1066-1075.

3 V. S. Anil Kumar and H. Ramesh, “Covering Rectilinear Polygons With Axis-Parallel Rectangles,” STOC 1999, New York, 445-454.

4 N. Bansal, D. Coppersmith, and B. Schieber. “Minimizing Setup and Beam-On Times in Radiation Therapy,” APPROX 2006, 27-38.

5 S. Bu, V. S. Lakshmanan, and R. T. Ng. MDL Summarization with Holes. In *VLDB ’05: Proceedings of the 31st international conference on Very Large Databases*, pages 433–444, VLDB Endowment, 2005.

6 E. Candes and T. Tao. Decoding By Linear Programming. In *IEEE Transactions on Information Theory* 51 (12), 2005, 4203-4215.

7 G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Diamond in the Rough: Finding Hierarchical Heavy Hitters in Multi-Dimensional Data. In *Proc. of ACM SIGMOD ’04*, Paris, France, 2004.

8 D. Donoho. For Most Large Underdetermined Systems of Linear Equations the Minimal ℓ^1-norm Solution is also the Sparsest Solution, In *Communications on Pure and Applied Mathematics* 59 (6), June 2006, 797-829.

9 A. Frieze and R. Kannan. Quick Approximation to Matrices and Applications. In *Combinatorica* 19 (2), 175-220, 1999.

10 C. Hurkens and A. Schrijver. “On the Size of Systems of Sets Every t of Which Have an SDR, With an Application to the Worst-Case Ratio of Heuristics for Packing Problems,” *SIAM J. Discrete Math.* 2(1), 1989, 68-72.

11 M. Karpinski and W. Schudy. Linear Time Approximation Schemes for the Gale-Berlekamp Game and Related Minimization Problems. In *STOC ’09: Proceedings of the 41st Annual ACM Symposium on Theory of Computing*, pages 313–322, New York, NY, USA, 2009. ACM.

12 V.S. Lakshmanan, R.T. Ng, C. Xing Wang, X. Zhou, and T. Johnson. The Generalized MDL Approach for Summarization. In *VLDB*, pages 766–777, 2002.

13 B. K. Natarajan. “Sparse Approximate Solutions To Linear Systems," *SIAM J. Comp.* 24(2), 1995, 227-234.

14 R. M. Roth and K. Viswanathan. On the Hardness of Decoding the Gale-Berlekamp Code. *IEEE Transactions on Information Theory*, 54(3):1050–1060, 2008.

15 S. Sarawagi. Explaining Differences in Multidimensional Aggregates. In *Proc. of the 25th International Conference on Very Large Databases (VLDB)*, pages 42–53, Scotland, UK, 1999.

16 Wikipedia page `http://en.wikipedia.org/wiki/Restricted_isometry_property`.

Unary negation*

Balder ten Cate[1] **and Luc Segoufin**[2]

1 **University of California, Santa Cruz**
`http://users.soe.ucsc.edu/~btencate`

2 **INRIA and ENS Cachan, LSV**
`http://www-rocq.inria.fr/~segoufin`

Abstract

We study fragments of first-order logic and of least fixed point logic that allow only unary negation: negation of formulas with at most one free variable. These logics generalize many interesting known formalisms, including modal logic and the μ-calculus, as well as conjunctive queries and monadic Datalog. We show that satisfiability and finite satisfiability are decidable for both fragments, and we pinpoint the complexity of satisfiability, finite satisfiability, and model checking. We also show that the unary negation fragment of first-order logic is model-theoretically very well behaved. In particular, it enjoys Craig interpolation and the Beth property.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Decidability, Logic, Unary negation

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.344

1 Introduction

Vardi [24] raised the question "why is modal logic so robustly decidable?". His explanation was that modal logic enjoys a combination of three properties, namely (i) the *tree model property* (if a sentence has a model, it has a model which is a tree), (ii) *translatability into monadic second-order logic* (MSO), and thereby into tree automata, and (iii) the *finite model property* (if there is a model, there is also a finite one). The decidability of (finite) satisfiability follows immediately from these three properties. The guarded fragment of first-order logic (GFO) [1] was subsequently proposed as a large fragment of first-order logic that generalizes modal logic while retaining these properties. It consists of FO formulas in which all quantifiers are "guarded". GFO has the tree-like model property (if a sentence has a model, it has a model of bounded tree width), it can be interpreted into MSO (each formula can be transformed into a tree automata recognizing a tree decomposition of its models of bounded tree width) and it has the finite model property [1, 15].

In this paper we provide another, orthogonal generalization of modal logic that enjoys the same nice properties. We introduce UNFO, a fragment of FO in which *negation* is restricted to formulas having only one free variable. UNFO is incomparable in term of expressive power to GFO but it generalizes modal logic, as well as other formalisms, such as conjunctive queries, that are not contained in GFO. We show that UNFO has the tree-like model property, interpretation into MSO and the finite model property. Hence UNFO is robustly decidable.

We also introduce UNFP, which extends UNFO with least and greatest monadic fixpoints, in the same way that the μ-calculus extends modal logic, and guarded fixpoint logic GFP

* Balder ten Cate has been funded partially by the ERC grant Webdam, agreement 226513, and partially by the NSF grant IIS-0905276.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 344–355
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

extends GFO [16]. UNFP generalizes the μ-calculus but also monadic Datalog and remains incomparable with GFP. It still has the tree-like model property and can be interpreted into MSO, but it no longer has the finite model property. Nevertheless, we show that finite satisfiability for UNFP is decidable (recall that the decidability of finite satisfiability for GFP is still open at the time of submission[1]). The satisfiability problem is 2ExpTime-complete, both for UNFO and for UNFP, both on arbitrary and finite structures.

We also study the model checking problem. In contrast with GFO, whose model checking problem is PTime-complete [7], we show that for UNFO it is complete for $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$, providing one of the few natural complete problems for that complexity class. For UNFP, model checking is hard for P^{NP} and contained in $\mathrm{NP}^{\mathrm{NP}} \cap \mathrm{coNP}^{\mathrm{NP}}$. The gap between the upper-bound and the lower-bound reflects a similar open problem for GFP and the μ-calculus where the model checking problems lies between PTime and NP $\cap$ coNP [7].

UNFO is not only computationally but also model-theoretically very well behaved. We characterize the expressive power of UNFO in terms of an appropriate notion of invariance, and we show that UNFO has the Craig Interpolation Theorem as well as the Projective Beth Property. Note that Craig Interpolation fails for GFO [18]. On trees, UNFO and UNFP correspond to well known formalisms.

Unary negation vs. guarded quantification As mentioned, UNFO and GFO are incomparable in term of expressive power. For instance the properties “some node lies on a cycle of length 4” and “every node lies on a cycle of length 4” (and their negations) are definable in UNFO but not in GFO (and not even in GFP). Conversely, the property “the binary relation R is contained in binary relation S” (and its negation) is definable in GFO but not in UNFO (and not even in UNFP). See Section 6 for more discussion. In spite of these differences, our proofs often follow similar strategies as proofs for GFO and GFP.

Due to space limitations many proofs are omitted or only sketched. They will appear in the journal version of this paper.

2 Preliminaries

We deal with relational structures. We assume given a relational schema providing a finite set of relation symbols and fixing an arity to each relation. A *model*, or *structure*, over a relational schema is a set, the *domain*, together with an interpretation of each relation symbol of the schema as a relation over the domain of the arity given by the schema. A model is said to be *finite* if its domain is finite. We assume familiarity with first-order logic, FO, and least fixpoint logic, FP, over relational structures. We use classical syntax and semantics for FO and FP. In particular we write $M \models \phi(\overline{u})$ for the fact that the tuple $\overline{u}$ of elements of the model M makes the FO-formula ϕ true on M.

2.1 UNFO and UNFP

We define *unary negation FO*, UNFO, as the fragment of FO given by the following grammar (where R is an arbitrary relation name from the underlying schema):

$$\phi :: \quad R(\overline{x}) \mid x = y \mid \phi \wedge \phi \mid \phi \vee \phi \mid \exists x \phi \mid \neg\varphi(x)$$

[1] Possibly recently solved [4].

where φ has no free variables besides (possibly) x. Throughout this paper, we will keep using the notation $\varphi(x)$ to indicate that a formula has at most one free variable. In other words, UNFO is the restriction of FO where negation is only allowed if the subformula has at most one free variable. In particular $x \neq y$ is not expressible in UNFO.

We say that a formula of UNFO is in UN-normal form if, in the syntax tree of the formula, every existential quantifier (except for the root) is either directly below another existential quantification, or the subformula starting with that quantifier has at most one free variable. It is immediate to see that each formula of UNFO can be turned into an equivalent one in UN-normal form in linear time by "pulling out existential quantifiers" as much as possible. For instance the formula $\exists x R(x) \wedge \exists x \neg(\exists y S(x,y))$ is not in UN-normal form but it is equivalent to $\exists x \exists x' \;\; R(x) \wedge \neg(\exists y S(x',y))$ which is in UN-normal form.

A formula of UNFO is said to be *of width k* if, when put in UN-normal form, it uses at most k variables. We denote by UNFO^k all UNFO formulas of width k.

In order to define UNFP we introduce extra unary predicates that will serve for computing unary fixpoints. We denote the unary predicates given by the relational schema using the letters $P, Q \ldots$ and the unary predicates serving for computing the fixpoints by $X, Y \ldots$. By UNFO($\overline{X}$) we mean UNFO defined over the schema extended with the unary predicates $\overline{X}$. In particular it allows formulas of the form $\neg\phi(x, \overline{X})$. UNFP is the extension of UNFO($\overline{X}$) by means of the following least fixpoint construction:

$$[\mathrm{LFP}_{X,x} \;\phi(X, \overline{X}, x)](y)$$

where X occurs only positively in ϕ. An analogous greatest fixed point operator is definable by dualization. Note that no first-order parameters (i.e., free variables in the body of ϕ other than the bound variable x) are permitted.

Since UNFP is a syntactic fragment of least fixed point logic LFP, we omit the definition of the semantics, cf. [19]. The definition of the normal form naturally extend to UNFP. As in the case of UNFO, a UNFP formula *has width k* if, when put in UN-normal form, it uses at most k variables. In particular, a formula of UNFP has width k if all the UNFO($\overline{X}$)-parts of its subformulas have width k. We denote by UNFP^k all UNFP formulas of width k.

The *negation depth* of a UNFO or UNFP formula will be an important parameter. It is the maximal nesting depth of negations in its syntax tree.

2.2 Logics that are contained in UNFO and UNFP

UNFO and UNFP generalize many known formalisms. We list here some well known logics that can be embedded into UNFO and UNFP.

Conjunctive queries and monadic datalog. A conjunctive query (CQ) is a query of the form $\exists x_1 \cdots x_n \; \tau_1 \wedge \cdots \wedge \tau_l$ where each τ_i is a positive atomic formula. Unions of conjunctive queries (UCQs) are contained in UNFO. Actually, UNFO can naturally be viewed as an extension of the UCQ language with unary negation. Similarly, monadic datalog (i.e., datalog queries in which all IDB relations are unary [12]) is contained in UNFP. As a matter of fact, if one allows the answer predicate of monadic datalog programs to have any arity, then monadic datalog corresponds precisely to the positive fragment of UNFP.

Modal logic and the μ-calculus. Modal logic is contained in UNFO. Indeed, the standard translation from modal logic to first-order logic produces first-order formulas that belong to UNFO. Similarly, the μ-calculus (and in fact the two-way μ-calculus) is contained in UNFP.

Unary conjunctive view logic. First-order unary-conjunctive-view logic (UCV) was introduced in [3] as a fragment of FO. A UCV query is an equality-free first-order formula over a signature consisting of unary predicates only, but where each of these unary predicates is in fact a view defined by a unary conjunctive query. UCV queries can be translated into UNFO, more precisely into the fragment of UNFO that has negation depth 1 (as follows from the fact that equality-free monadic FO admits quantifier elimination, cf. [17]).

The temporal logic CTL*(X). CTL*(X) is the fragment of CTL* in which only the modality X ("next") is allowed (For the definition of the semantics, see [11]). CTL*(X) is a fragment of UNFO. The model checking problem for CTL*(X) is known to be complete for the complexity class $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$ [23]. We will show that, in fact, the model checking problem for full UNFO is $\mathrm{P}^{\mathrm{NP}[O(\log^2 n)]}$-complete.

Core XPath. On XML trees with all axes, it is known that *Core XPath* = FO^2 for unary queries while *Core XPath* = *UCQ-over-*FO^2*-unary-predicates* for binary queries [20]. It turns out that UNFO has the same expressive power as Core XPath, both for unary and for binary queries [10] and therefore UNFO characterizes Core XPath in a more uniform way. Similarly, Regular XPath embeds into UNFP.

Data tree patterns. In [13], *Boolean combinations of tree patterns* are studied as a query language for data trees (or, XML documents containing data). If we represent data trees by relational structures, then tree patterns can be seen as conjunctive queries. Therefore the Boolean combinations of tree patterns studied in [13] can be seen as a fragment of UNFO.

Description logics. The *conjunctive query answering problem for description logics* can be reduced to the UNFO entailment problem. Take for example the basic description logic $\mathcal{ALC}$. The conjunctive query answering problem for $\mathcal{ALC}$ is the following problem (cf. [2] for basic terminology): *Given a TBox T, an ABox A consisting of atomic formulas speaking about individuals (constant symbols) $c_1 \ldots c_n$, a conjunctive query $q(x_1, \ldots, x_k)$, and a tuple of constants $(c_{i_1}, \ldots, c_{i_k})$, is $(c_{i_1}, \ldots, c_{i_k})$ an answer to q in every model of $T \cup A$?*

It is easy to see that this is equivalent to the validity of the UNFO-entailment

$$\phi_T \wedge \bigwedge A[c_1/x_1, \ldots, c_n/x_n] \quad \models \quad q(x_{i_1}, \ldots, x_{i_k})$$

where ϕ_T is the UNFO-translation of T. It follows that conjunctive query answering for $\mathcal{ALC}$ is decidable and has the finite model property (cf. Remark 3). The same argument works not only for $\mathcal{ALC}$ but for any description logic whose TBoxes can be expressed in UNFP. Moreover, the argument works not only for conjunctive queries, but to any class of queries expressible in UNFP.

3 Model theory

In this section we give results about the expressive power of UNFO and UNFP, and we show that UNFO has Craig Interpolation Theorem and the Projective Beth Property. We only state the results here, the proofs will appear in the journal version of this paper.

Invariance for UN-bisimulations We define a game that captures model indistinguishability, and we use it to characterize the expressive power of UNFO and UNFP. The game is as follows: the two players maintain a single pair (a, b) of objects from the two structures. A move of Abelard consists of choosing a set X of points in one of the two structures. Then Eloise responds with a homomorphism from that set into a set of points in the other structure, where the homomorphism maps a to b (respectively b to a) if a (respectively b) belongs to the set X. Finally, Abelard picks a pair from the homomorphism and the players continue with that pair. The game is parametrized by the size of the sets chosen by Abelard at each round.

Equivalently, we can present the game in terms of a back-and-forth system:

▶ **Definition 1.** Let M, N be two structures. A UN-bisimulation (of width $k \geq 1$) is a binary relation $Z \subseteq M \times N$ such that the following hold for every pair $(a, b) \in Z$:

- For every finite set $X \subseteq \text{dom}(M)$ (with $|X| \leq k$) there is a partial homomorphism $h : M \to N$ whose domain is X, such that $h(a) = b$ if $a \in X$, and such that every pair $(a', b') \in h$ belongs to Z.
- Likewise in the other direction, where $X \subseteq \text{dom}(N)$.

We write $M \approx_{\text{UN}} N$ if there is a non-empty UN-bisimulation between M and N, and we write $M \approx^k_{\text{UN}} N$ if there is a non-empty UN-bisimulation of width k between M and N.

It is not difficult to see that UN-bisimulation implies UNFP-indistinguishability.

▶ **Proposition 1.** *For any $k \geq 1$, if $M \approx^k_{UN} N$ then M and N satisfy the same sentences of* UNFP^k*. In particular, if $M \approx_{UN} N$ then M and N satisfy the same sentences of* UNFP*.*

A similar invariance property holds for formulas with free variables. For simplicity, we only state a version of the result without reference to the width of formulas. Let a *UN-homomorphism* $h : M \to N$ be a homomorphism with the property that $M, a \approx_{\text{UN}} N, h(a)$ for all $a \in \text{dom}(M)$. We write $M, \overline{a} \to_{\text{UN}} N, \overline{b}$ if there is a UN-homomorphism $h : M \to N$ such that $h(\overline{a}) = \overline{b}$. Then we have the following:

▶ **Proposition 2.** *If $M, \overline{a} \to_{\text{UN}} N, \overline{b}$ and $M \models \phi(\overline{a})$ then $N \models \phi(\overline{b})$, for all* UNFP*-formulas $\phi(\overline{x})$.*

Tree-like model property and finite model property From the invariance for UN-bisimulation it follows by a standard infinite unraveling argument that UNFP has the tree-like model property. A more involved partial unraveling, using back-edges in order to keep the structure finite, can also be used to show that UNFO has the finite model property. We will not give the details of these constructions, as it turns out that both results will follow from the material presented in Section 4.

▶ **Theorem 2.** *Every satisfiable* UNFO *formula has a finite model.*

▶ **Theorem 3.** *Every satisfiable* UNFP *formula of width k has a model of tree-width $k - 1$.*

Note, that UNFP does *not* have the finite model property. This follows from the fact that UNFP contains the two-way μ-calculus which is known to lack the finite model property [8]. Indeed, if $\max(x)$ is shorthand for $\neg\exists y(E(x, y))$, then the formula

$$\exists x \max(x) \vee \exists x[\text{GFP}_{X,y}\ \exists z(Xz \wedge E(z, y))](x)$$

is valid on finite structures (if a finite structure has no maximal elements, it must contain a cycle, and hence an infinite backward path) but it is false in the infinite structure $(\mathbb{N}, \mathit{suc})$.

Characterization We have seen in Proposition 1 that UNFO sentences are first-order formulas that are closed under $\approx_{\text{UN}}$-equivalence. It turns out that the converse is also true. Indeed, in the same way that bisimulation-invariance characterizes modal logic [6, 22] and guarded bisimulation-invariance characterizes the guarded fragment of FO [1], we will see that $\approx_{\text{UN}}$-invariance characterizes UNFO. We state this result over arbitrary models. We believe the result can also be proved over finite structures. We postpone this issue to the journal version of this paper.

Call a FO sentence $\approx_{UN}$*-invariant* if for all structures $M \approx_{\text{UN}} N$, we have $M \models \phi$ iff $N \models \phi$, and define $\approx^k_{\text{UN}}$-invariance similarly. Then

▶ **Theorem 4.** UNFO *is the* $\approx_{UN}$*-invariant fragment of* FO*, and for all* $k \geq 1$*,* UNFO^k *is the* $\approx^k_{UN}$*-invariant fragment of* FO *(on arbitrary structures)*

The result above applies to sentences. A similar characterization can be obtained for formulas with free variables, using UN-homomorphisms instead of UN-bisimulations.

▶ **Theorem 5.** UNFO *formulas (with free variables) form the fragment of* FO *that is preserved under UN-homomorphisms.*

Craig interpolation and Beth definability We conclude the list of nice model-theoretic properties of UNFO by showing that it has Craig Interpolation Theorem and the Projective Beth Property. In fact, we can show strong versions of these results, which take into account also the width of formulas. This is remarkable, given that both Craig Interpolation and the Beth Property fail for the k-variable fragment of first-order logic, for all $k > 1$. Moreover, the results presented in this section hold both on arbitrary structures and on finite structures.

For all UNFO-formulas $\phi(\overline{x}), \psi(\overline{x})$, we write $\phi \models \psi$ to express that the first-order formula $\forall\overline{x}(\phi \to \psi)$ (which is not necessarily a UNFO-formula) is valid.

▶ Remark. For all UNFO-formulas $\phi(\overline{x}), \psi(\overline{x})$, $\phi \models \psi$ holds (on finite structures) if and only if the formula

$$\exists\overline{x}(\phi \wedge \bigwedge_i P_i(x_i)) \wedge \neg\exists\overline{x}(\psi \wedge \bigwedge_i P_i(x_i))$$

is not satisfiable (on finite structures). Hence, all results we prove for sentences (e.g., the complexity of satisfiability, the finite model property, etc.) all apply to entailment as well.

Below, we state and prove our results for arbitrary structures, but the analogous results for finite structures follow by Theorem 2 and Remark 3.

▶ **Theorem 6.** UNFO^k *has Craig interpolation: for all* $k \geq 1$ *and for every pair of* UNFO^k*-formulas* $\phi(\overline{x}), \psi(\overline{x})$ *in the same free variables such that* $\phi \models \psi$*, there is a* UNFO^k*-formula* $\chi(\overline{x})$ *over the common vocabulary of* ϕ *and* ψ *such that* $\phi \models \chi$ *and* $\chi \models \psi$*.*

As usual, this Craig interpolation theorem implies a Beth definability theorem. Let Σ be a UNFO-theory in a signature σ and let $R \in \sigma$ and $\tau \subseteq \sigma$. We say that Σ *implicitly defines* R in terms of τ if for all τ-structures M and for all σ-expansions M_1, M_2 of M satisfying Σ, we have that $R^{M_1} = R^{M_2}$. We say that a formula $\phi(\overline{x})$ in signature τ is an *explicit definition* of R relative to Σ if $\Sigma \models \forall\overline{x} \ \ (R\overline{x} \leftrightarrow \phi(\overline{x}))$. Note that the formula $\forall\overline{x} \ \ (R\overline{x} \leftrightarrow \phi(\overline{x}))$ is itself not necessarily a UNFO-formula, but this is irrelevant.

▶ **Theorem 7.** UNFO *has the Projective Beth property: whenever a* UNFO*-theory* Σ *in a signature* σ *implicitly defines a* k*-relation* R *in terms of a signature* $\tau \subseteq \sigma$*, then there is a* UNFO*-formula in signature* τ *that is an explicit definition of* R *relative to* Σ*. Moreover, if* Σ *belongs to* UNFO^k *(*$k \geq 1$*), then the explicit definition can be found in* UNFO^k *as well.*

4 Satisfiability

In this section, we show that the satisfiability problem for UNFP and for UNFO is 2ExpTime-complete, both on arbitrary structures and on finite structures. The lower-bound holds already for UNFO^3 over finite trees. Note that this is in contrast with GFO whose complexity drops from 2ExpTime-complete to ExpTime-complete when the arity of relations is bounded [15]. The upper-bound is obtained by a reduction to the two-way modal μ-calculus, whose satisfiability and finite satisfiability problems are known to be ExpTime-complete [8]. Given a formula φ of UNFP we construct in exponential time a formula φ^* in the μ-calculus such that φ has a (finite) model iff φ^* has a (finite) model. The construction of a finite model of φ from a finite model of φ^* uses a result from [21], which implies that we can restrict attention to locally acyclic structures (i.e., structures that do not contain short cycles). The same reduction to the two-way modal μ-calculus allows us to prove the finite model property of UNFO and the tree-like model property of UNFP.

We describe the reduction from φ to φ^* in two parts. In the first one we consider only a special case of UNFP formulas that we call *simple* where, intuitively, each conjunctive query inside φ is a single atomic formula. The construction of φ^* is then polynomial. In a second part we show how the general case reduces to this one (with an exponential blow-up).

4.1 Simple UNFP formulas

We first consider a fragment of UNFP, which we call *simple* UNFP (sUNFP). It is a common fragment of UNFP and GFP, which embeds the two-way μ-calculus. The syntax of sUNFP is given by the following grammar (recall that we use the notation $\phi(x)$ to indicate that a formula has no first-order free variables besides possibly x, but may contain some monadic second order free variables):

$$\begin{aligned}\phi(x) ::= \quad & P(x) \mid X(x) \mid \phi(x) \wedge \phi(x) \mid \phi(x) \vee \phi(x) \mid \neg\phi(x) \mid [\mathrm{LFP}_{X,y}\phi(y)](x) \mid \\ & \exists y_1 \dots y_n (R(y_1 \dots y_n) \wedge y_i = x \wedge \bigwedge_{j \in \{1 \dots n\} \setminus \{i\}} \phi_j(y_j)) \ \mid \ \exists x \phi(x)\end{aligned}$$

Note that all formulas generated by this inductive definition have at most one free variable. We denote by sUNFO the first-order (fixed point free) fragment of sUNFP.

We need the following notions. The *incidence graph* $inc(M)$ of a structure M is the bi-partite graph containing facts of M and elements of M, and with an edge between a fact and an element if the element occurs in the fact. We say that a structure M is l-acyclic, for $l \geq 1$, if $inc(M)$ has no cycle of length less than $2l$, and no element of M occurs twice in the same fact. We call a structure acyclic, if it is l-acyclic for all l (i.e., the incidence graph is acyclic and no element occurs in the same fact twice).

A simple formula is essentially a formula of the two-way μ-calculus with navigation through arbitrary relations instead of just binary relations. Based on a simple coding of relations of arbitrary arity using binary relations we can transform a simple formula into a formula of the μ-calculus and obtain:

▶ **Proposition 3.**

1. *The satisfiability problem for* sUNFP *is* ExpTime-*complete, both on arbitrary structures and on finite structures.*
2. *If a* sUNFP *formula has a model, it has an acyclic model*
3. *If a* sUNFP *formula has a finite model, then it has a l-acyclic finite model,* $\forall l \geq 1$.
4. sUNFO *has the finite model property.*

4.2 Arbitrary UNFP-formulas

A formula of UNFO is said to be in *disjunctive* UN-normal form if it is a disjunction of formulas that are in UN-normal form and in which only unary disjunction, of the form $\phi(x) \vee \psi(x)$, is used. It is immediate to see that every formula of UNFO can be turned into an equivalent (but possibly exponentially longer) one in disjunctive UN-normal form. It turns out that the parameters that will occur in the exponent of the reduction described below (for instance the width) are not affected when going from an arbitrary formula to one in disjunctive normal form. Hence we can now fix an arbitrary UNFP-formula ϕ in disjunctive UN-normal form without loss of generality.

Step 1: Simplifying assumptions (without loss of generality)

1. ϕ is a sentence (all free variables can be existentially quantified, cf. also Remark 3).
2. Every subformula of the form $\exists\overline{z}\psi(y,\overline{z})$ with $\overline{z} = z_1 \dots z_n$ is more precisely of the form

$$\exists\overline{z}(\tau(\overline{z}) \wedge z_i = y \wedge \bigwedge_{j\in\{1\dots n\}\setminus\{i\}} \phi_j(z_j)) \qquad \text{or} \qquad \exists\overline{z}(\tau(\overline{z}) \wedge \bigwedge_{j\in\{1\dots n\}} \phi_j(z_j))$$

for some $i \leq n$, where $\tau(\overline{z})$ is a conjunction of relational atomic formulas (no equalities, those can be eliminated by identifying the respective quantified variables).

Step 2: Collecting subformulas and neighborhood types We denote by $\textsc{subf}_\phi$ the set of all subformulas $\psi(y)$ of ϕ that have one free first-order variable. Next, we collect all conjunctive queries occurring in ϕ, viewing each as describing a neighborhood type. For any subformula of ϕ of the form

$$\exists\overline{z}(\tau(\overline{z}) \wedge z_i = y \wedge \bigwedge_{j\in\{1\dots n\}\setminus\{i\}} \phi_j(z_j)) \qquad \text{or} \qquad \exists\overline{z}(\tau(\overline{z}) \wedge \bigwedge_{j\in\{1\dots n\}} \phi_j(z_j)) \ ,$$

we call $\tau(\overline{z})$ a *neighborhood type*. We denote the set of neighborhood types in ϕ by $\textsc{ntypes}_\phi$.

Step 3: Stitching neighborhood types together. We now consider structures that are "stitched together" from copies of the neighborhood types in $\textsc{ntypes}_\phi$. To make this precise, we introduce for each neighborhood type $\tau(z_1,\dots,z_n) \in \textsc{ntypes}_\phi$ an n-ary relation symbol R_τ. A structure in the signature consisting of these new relations will be called a *stitch diagram*. Each stitch diagram M gives rise to a *stitching* $M^\times$, which is the structure (over the same domain of M) obtained by replacing each R_τ-fact with a copy of the neighborhood type τ, for all $\tau \in \textsc{ntypes}_\phi$. Notice that, when going from M to $M^\times$, the distance between nodes can only increase (by second simplifying assumption, τ does not contain equalities hence no nodes are merged during the process).

At this point, our basic strategy for reducing UNFP to sUNFP should be clear: we will produce an sUNFP-formula to describe stitch diagrams whose stitchings satisfy a certain desired UNFP formula. In the rest of this section, we work out the details of this strategy.

It is important to realize that, even if a stitch diagram M does not contain an atomic fact $R_\tau(\overline{a})$, it may still be the case that $M^\times \models \tau(\overline{a})$. In this case we say that the fact $R_\tau(\overline{a})$ is *implicit* in M. For example, this could happen if $M \models R_{\tau'}(\overline{a})$ and τ is homomorphically contained in τ'. The following claim gives us a handle on when this phenomenon may occur. For any $\tau \in \textsc{ntypes}_\phi$, we denote by $|\tau|$ the number of atomic formulas in τ. We write $N \subseteq M$ if N is a not-necessarily-induced substructure of M.

▶ **Claim 1.** *If $R_\tau(\overline{a})$ is implicit in a stitch diagram M then there is an $N \subseteq M$ containing at most $|\tau|$ many facts, such that $R_\tau(\overline{a})$ is already implicit in N. Moreover N is connected whenever τ is.*

Proof. We need at most one fact of M to account for each atom in $\tau(\overline{a})$. ◀

Let $l = \max_{\tau \in \textsc{ntypes}_\phi} |\tau|$. We will restrict attention to stitch diagrams M that are l-acyclic. By Item (3) of Proposition 3 this is without loss of generality. This implies that every $N \subseteq M$ containing at most l facts is fully acyclic. The importance of the above claim, then, shows in two facts: (i) intuitively, there are finitely many reasons why a fact may be implicit in M, and (ii) each of these reasons is acyclic, and hence can be described in sUNFP as we will see.

Step 4: The translation from UNFP to sUNFP Let $\psi(y)$ be any subformula of ϕ with at most one free variable. By induction on the structure of $\psi(y)$ we construct a sUNFP formula $\psi'(y)$ such that, assuming $\overline{X}$ are the free monadic variables of ψ, for all l-acyclic M, all $a \in M$, and all sets $\overline{S}$ of elements of M, $M, \overline{S} \models \psi'(a)$ iff $M^\times, \overline{S} \models \psi(a)$.

The inductive definition commutes with all Boolean operator and with the LFP operator. Fix now any $\tau(\overline{z}) \in \textsc{ntypes}_\phi$ with $\overline{z} = z_1, \ldots, z_n$, fix an $i \leq n$, and fix a sequence of formulas $\psi_1, \ldots, \psi_{i-1}, \psi_{i+1}, \ldots, \psi_n \in \textsc{subf}_\phi$ and assume ϕ is of the form:

$$\psi(y) \;:=\; \exists \overline{z}(\tau \wedge z_i = y \wedge \bigwedge_{j \in \{1 \ldots n\} \setminus \{i\}} \psi_j(z_j)) \; .$$

(the argument if ψ is of the form $\exists \overline{z}(\tau \wedge \bigwedge_{j \in \{i \ldots n} \psi_j(z_j))$ is similar. Note that these two cases also account for the base of the induction, if we let $|\overline{z}| = 1$).

By induction we already have constructed sUNFP formulas $\psi'_1, \ldots, \psi'_{i-1}, \psi'_{i+1}, \ldots, \psi'_n$ corresponding to $\psi_1, \ldots, \psi_{i-1}, \psi_{i+1}, \ldots, \psi_n$.

We are interested in detecting in M how a node in $M^\times$ may come to satisfy ψ. We will construct a sUNFP formula that lists all the cases in M that make this happen. It clearly suffices to consider each connected component of τ at a time. Hence by Claim 1 it only depends on a small neighborhood of x in M. The formula will then be essentially a long disjunction, where each disjunct corresponds to the description of a small neighborhood of M in which τ is implicitly satisfied by a tuple of nodes satisfying in addition the formulas ψ_j. Note that since we assume M to be l-acyclic, these small substructures are all acyclic, which will make it possible to describe them by an (existential) formula of sUNFP.

More precisely, consider any connected acyclic stitch diagram N containing at most l facts, and any homomorphism $h : \tau \to N^\times$. We now construct an sUNFP formula $\chi_{\psi,N,h}(y)$ that describes N (existentially positively) from the point of view of $h(z_i)$, and expressing also that each $h(z_j)$ satisfies ψ_j. In other words, we view N as a tree rooted in $h(z_i)$ and the formula describes that tree from top to bottom. We construct the desired sUNFP formula by induction on $|N|$.

Assume N is a tree whose root element is $N_0 = R_\tau(a_1, \cdots, a_n)$ and with several subtrees $N_1, N_2 \cdots$ (the base case where N is a single node is treated similarly). Notice that by l-acyclicity it follows that for all $m > 0$, N_0 and N_m share at most one element, say a_{β_m}. For $m \geq 0$, let h_m be the restriction of h to N_m. Finally assume that h_0 maps z_i to a_{α_i}. The desired formula $\chi_{\phi,N,h}(y)$ is then:

$$\exists z_1 \cdots z_{\alpha_i - 1} z_{\alpha_i + 1} \cdots z_n \Big(R_\tau(z_1 \cdots z_{\alpha_i - 1} y z_{\alpha_i + 1} \cdots z_n) \;\wedge\; \bigwedge_{j \neq i, h_0(z_j) = a_{\alpha_j}} \psi'_j(z_{\alpha_j}) \;\wedge\; \bigwedge_m \chi_{\psi,N_m,h_m}(z_{\beta_m}) \Big)$$

Finally $\psi'(y)$ is the disjunction, for each N and h as above, of the formulas $\chi_{\psi,N,h}(y)$.

It follows from the construction that, for all l-acyclic stitch diagrams M, $M \models \psi'$ if and only if $M^\times \models \psi$. This shows that, if ψ' is satisfiable, then so is ψ. Conversely, it is easy to construct, from a model M of ψ, a model M' of ψ' (indeed, it suffices to take the domain M and to populate the relations R_τ with all tuples satisfying τ in M. Therefore, ψ is satisfiability (on finite structures) if and only if ψ' is.

A careful analysis of the complexity of the above translation yields:

▶ **Theorem 8.** *The satisfiability problem for* UNFP *is in 2*ExpTime*, both on arbitrary structures and on finite structures.*

Notice now that when starting with a formula of UNFO we obtain a formula of sUNFO. In view of Item (4) of Proposition 3, this immediately implies the finite model property of UNFO and proves Theorem 2. Similarly, it follows from Item (2) of Proposition 3 that UNFP has the tree-like model property. Indeed if a stitch diagram M is acyclic, then $M^\times$ has tree-width at most $k-1$, where k is the width of the formula. This proves Theorem 3.

The complexity result of Theorem 8 is tight:

▶ **Proposition 4.** *There is a fixed finite signature such that the satisfiability problem for* UNFO *is 2*ExpTime*-hard, both on arbitrary structures and on finite structures.*

The lower bound can be shown on arbitrary structures, on finite trees, and on any class in-between. The proof uses formulas that have width 3 and negation depth 2. For width 2 we can actually show that satisfiability of UNFO^2 is NExpTime-complete. For negation depth 1 it turns out to be NP$^{\mathrm{NP}}$-complete.

5 Model Checking

In this section we study the model checking complexity of UNFO and UNFP. We are concerned here with the *combined complexity* of the model checking problem, where the input consists of a formula and a structure. It was already observed in [10] that the model checking problem for UNFO is in P$^{\mathrm{NP}}$. Here, we show that the problem is in fact P$^{\mathrm{NP}[O(\log^2 n)]}$-complete, and that the model checking problem for UNFP is in NP$^{\mathrm{NP}}$ $\cap$ coNP$^{\mathrm{NP}}$. The complexity class P$^{\mathrm{NP}}$ consists of all problems that are computable by a Turing machine running in time polynomial in the size of its input, where the Turing machine, at any point during its computation, can ask yes/no queries to an NP oracle, and take the answers of the oracle into account in subsequent steps of the computation (including subsequent queries to the NP oracle). Analogously we can define the complexity classes NP$^{\mathrm{NP}}$ and coNP$^{\mathrm{NP}}$. The complexity class P$^{\mathrm{NP}[O(\log^2 n)]}$ is defined in the same way as P$^{\mathrm{NP}}$, except that the number of yes/no queries that can be asked to the NP oracle is bounded by $O(\log^2(n))$, where n is the size of the input. Similarly P$^{\mathrm{NP}[O(\log n)]}$ restricts the number of yes/no queries to $O(\log(n))$. We refer to [9, 14, 23, 25] for the precise definitions and proprieties of these oracle complexity classes.

▶ **Theorem 9.** *The model checking problem for* UNFO *is* P$^{\mathrm{NP}[O(\log^2 n)]}$*-complete.*

The lower bound follows from the fact that UNFO embeds CTL*(X), since the model checking problem for CTL*(X) is already known to be P$^{\mathrm{NP}[O(\log^2 n)]}$-complete [23]. The upper-bound argument is more involved. It uses $1 \times M$ *TB-trees* [23], a model for P$^{\mathrm{NP}[O(\log^2 n)]}$ computations based on tree-shaped circuits containing SAT tests. The argument relies on the fact that UNFO formulas can be viewed as such tree-shaped circuits, and it breaks down if subformula sharing is allowed. Indeed for a simple extension of UNFO with a let construct of the form let $b = \phi$ in ψ the model checking problem can be shown to be complete for P$^{\mathrm{NP}}$.[2]

Recall that the *negation depth* of a UNFO formula is the maximal nesting depth of negations in its syntactic tree. We can show the following result:

[2] Here, b is a Boolean variable and ϕ a sentence; let $b = \phi$ in ψ can be viewed as a succinct notation for the formula obtained by replacing all free occurrences of b in ψ by ϕ.

▶ **Theorem 10.** *For any $l > 0$, the complexity of the model checking problem for* UNFO *formula of negation depth $\leq l$ is* $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$*-complete. The lower bound holds even when the structure is fixed.*

The upper bound is obtained by induction on l using a naive bottom-up evaluation algorithm. Each level requires one series of parallel calls to the NP-oracle, hence the result when l is a constant. The lower-bound proof uses the fact that every problem in $\mathrm{P}^{\mathrm{NP}[O(\log n)]}$ admits a PTime truth-table reduction to a problem in NP [9]. The proof will be detailed in the journal version of this paper.

Finally, we turn to the complexity of the model checking of UNFP.

▶ **Theorem 11.** *The* UNFP *model checking problem is in* $\mathrm{NP}^{\mathrm{NP}} \cap \mathrm{coNP}^{\mathrm{NP}}$ *and* P^{NP}*-hard.*

The upper bound is proved using the obvious extension of the known NP∩coNP algorithm for the model checking of the μ-calculus, using the NP-oracle for solving the unary conjunctive queries. The lower bound will be detailed in the journal version of this paper.

6 Discussion

Trees Over trees, UNFO and UNFP correspond to well known formalisms. We have already mentioned that on XML trees UNFO captures Core XPath and it is not hard to see that UNFP captures the regular languages. When only the child relation of the tree is present, definability in UNFO correspond to "Locally Testability" while UNFP defines the bisimulation invariant regular languages.

Undecidable extensions Our results show that UNFO and UNFP are well behaved logics. One may ask if there are extensions that are still well behaved. Inequalities are a minimal form of negation not supported by UNFO. Unfortunately, extending UNFO with inequalities leads to undecidability. Let us denote by $\mathrm{UNFO}^{\neq}$ the extension of UNFO with inequalities, and with $\mathrm{UNFO}^{\neg}$ the extension of UNFO with negative relational atomic formulas. Recall that a fragment of first-order logic is called a *conservative reduction class* if there a computable map from arbitrary first-order formulas to formulas in the fragment, which preserves (un)satisfiability as well as finite (un)satisfiability.

▶ **Theorem 12.** $\mathrm{UNFO}^{\neq}$ *and* $\mathrm{UNFO}^{\neg}$ *are conservative reduction classes, and hence undecidable for satisfiability on arbitrary structures and on finite structures.*

Also, in the fixed point case, one can wonder whether the restriction to *monadic* least fixed-points was necessary. Indeed, this question naturally arises since it is known that the *guarded fragment* of first-order logic is decidable even when extended with (guarded) fixed point operators of arbitrary arity. However, it turns out that in our setting allowing non-monadic fixed points operators makes the logic undecidable.

▶ **Theorem 13.** *The extension of* UNFP *with non-monadic fixed point operators is undecidable for satisfiability on arbitrary structures and on finite structures.*

Further work We have already mentioned that UNFO and GFO are incomparable. It would be nice to come up with a logic that generalizes both UNFO and GFO. A step in this direction is the recent work of [5] showing that (finite) satisfiability of a Boolean combination of guarded and CQ formulas is decidable. With Vince Bárány we are currently investigating the *guarded negation fragment* of FO which allows negations of the form $R(\overline{x}) \wedge \neg\phi(\overline{x})$. This fragment, and its fixed point extension, generalize both UNFO and GFO, while apparently retaining their good properties, including robust decidability.

References

1 H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.

2 F. Baader, D. Calvanese, D.L. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

3 J. Bailey, G. Dong, and A.W. To. Logical queries over views: Decidability and expressiveness. *Transactions on Computational Logic*, 11(2):8, 2010.

4 V. Bárány and M. Bojańczyk. Personal communication.

5 V. Bárány, G. Gottlob, and M. Otto. Querying the guarded fragment. In *Proc. of Logic in Computer Science (LICS)*, 2010.

6 J. van Benthem. *Modal Logic and Classical Logic*. Bibliopolis, 1983.

7 D. Berwanger and E. Grädel. Games and model checking for guarded logics. In *LPAR*, pages 70–84, 2001.

8 M. Bojańczyk. Two-way alternating automata and finite models. In *Intl. Coll. on Automata, Languages and Programming (ICALP)*, pages 833–844, 2002.

9 S.R. Buss and H. Louise. On truth-table reducibility to sat. *Inf. Comput.*, 91(1):86–102, 1991.

10 B. ten Cate and M. Marx. Navigational XPath: calculus and algebra. *SIGMOD Record*, 36(2):19–26, 2007.

11 E.M. Clarke, O. Grumberg, and D.A. Peled. *Model checking*. MIT Press, 1999.

12 S. Cosmadakis, H. Gaifman, P. Kanellakis, and M. Vardi. Decidable optimization problems for database logic programs. In *Proc. of ACM Symposium on Theory of Computing (STOC)*, 1988.

13 C. David. *Analyse de XML avec données non-bornées*. PhD thesis, Université Paris Diderot - Paris 7 (LIAFA), 2009.

14 G. Gottlob. NP trees and Carnap's modal logic. *Journal of the ACM*, 42(2):421–457, 1995.

15 E. Grädel. Why are modal logics so robustly decidable? In *Current Trends in Theoretical Computer Science*, pages 393–408. 2001.

16 E. Grädel and I. Walukiewicz. Guarded fixed point logic. In *LICS*, pages 45–54, 1999.

17 W. Hodges. *Model Theory*. Cambridge University Press, 1993.

18 E. Hoogland and M. Marx. Interpolation and definability in guarded fragments. *Studia Logica*, 70(3):373–409, 2002.

19 L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.

20 M. Marx and M. de Rijke. Semantic characterizations of navigational XPath. *SIGMOD Record*, 34(2):41–46, 2005.

21 M. Otto. Bisimulation invariance and finite models. In *Lecture Notes in Logic, Logic Colloquium 2002*, pages 276–298. 2006.

22 E. Rosen. Modal logic over finite structures. *Journal of Logic, Language, and Computation*, 6(4):427–439, 1997.

23 P. Schnoebelen. Oracle circuits for branching-time model checking. In *International Conference on Automata, Languages and Programming*, pages 187–187, 2003.

24 M.Y. Vardi. Why is modal logic so robustly decidable? In *Descriptive Complexity and Finite Models*, pages 149–184, 1996.

25 K.W. Wagner. More Complicated Questions about Maxima and Minima, and some Closures of NP. *Theoretical Computer Science*, 51(1-2):53 – 80, 1987.

First-order Fragments with Successor over Infinite Words*

Jakub Kallas[1], Manfred Kufleitner[2], and Alexander Lauser[2]

1 **ENS Cachan, France**
jakub.kallas@gmail.com

2 **University of Stuttgart, FMI, Germany**
kufleitner@fmi.uni-stuttgart.de
lauser@fmi.uni-stuttgart.de

Abstract

We consider fragments of first-order logic and as models we allow finite and infinite words simultaneously. The only binary relations apart from equality are order comparison $<$ and the successor predicate $+1$. We give characterizations of the fragments $\Sigma_2 = \Sigma_2[<, +1]$ and $\mathrm{FO}^2 = \mathrm{FO}^2[<, +1]$ in terms of algebraic and topological properties. To this end we introduce the factor topology over infinite words. It turns out that a language L is in $\mathrm{FO}^2 \cap \Sigma_2$ if and only if L is the interior of an FO^2 language. Symmetrically, a language is in $\mathrm{FO}^2 \cap \Pi_2$ if and only if it is the topological closure of an FO^2 language. The fragment $\Delta_2 = \Sigma_2 \cap \Pi_2$ contains exactly the clopen languages in FO^2. In particular, over infinite words Δ_2 is a strict subclass of FO^2. Our characterizations yield decidability of the membership problem for all these fragments over finite and infinite words; and as a corollary we also obtain decidability for infinite words. Moreover, we give a new decidable algebraic characterization of dot-depth 3/2 over finite words.

Decidability of dot-depth 3/2 over finite words was first shown by Glaßer and Schmitz in STACS 2000, and decidability of the membership problem for FO^2 over infinite words was shown 1998 by Wilke in his habilitation thesis whereas decidability of Σ_2 over infinite words is new.

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.4.3 Formal Languages.

Keywords and phrases infinite words, regular languages, first-order logic, automata theory, semigroups, topology

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.356

1 Introduction

The dot-depth hierarchy of star-free languages $\mathcal{B}_n$ for $n \in \mathbb{N} + \{1/2, 1\}$ over finite words has been introduced by Brzozowski and Cohen [5]. Later, the Straubing-Thérien $\mathcal{L}_n$ hierarchy has been considered [20, 23] and a tight connection in terms of so-called wreath products was discovered [19, 21]. It is known that both hierarchies are strict [4] and that they have very natural closure properties [5, 18]. Effectively determining the level n of a language in the dot-depth hierarchy or the Straubing-Thérien hierarchy is one of the most challenging open problems in automata theory. So far, the only decidable classes are $\mathcal{B}_n$ and $\mathcal{L}_n$ for $n \in \{1/2, 1, 3/2\}$, see e.g. [17] for an overview and [10] for level $\mathcal{B}_{3/2}$.

Thomas showed that there is a one-to-one correspondence between the quantifier alternation hierarchy of first-order logic and the dot-depth hierarchy [25]. This correspondence

* The last two authors acknowledge the support by the German Research Foundation (DFG), grant DI 435/5-1.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 356–367
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

holds if one allows $[<, +1, \min, \max]$ as a signature (we always assume that we have equality and predicates for labels of positions; in order to simplify notation, these symbols are omitted here). The same correspondence between the Straubing-Thérien hierarchy and the quantifier alternation hierarchy holds, if we restrict the signature to $[<]$, cf. [18]. In particular, all decidability results for the dot-depth hierarchy and the Straubing-Thérien hierarchy yield decidability of the membership problem for the respective levels of the quantifier alternation hierarchy.

The intersection $\Delta_2[<] = \Sigma_2[<] \cap \Pi_2[<]$ of the language classes $\Sigma_2[<]$ and $\Pi_2[<]$ of the quantifier alternation hierarchy over finite words has a huge number of different characterizations, see [22] for an overview. One of them turns out to be the first-order fragment $\mathrm{FO}^2[<]$ where one can use (and reuse) only two variables [24]. The fragment $\mathrm{FO}^2[<]$ is a natural restriction since three variables are already sufficient to express any first-order language over finite and infinite words [12]. Using the wreath product principle [21], one can extend $\Delta_2[<] = \mathrm{FO}^2[<]$ to $\Delta_2[<, +1] = \mathrm{FO}^2[<, +1]$, see e.g. [14]. Decidability of $\mathrm{FO}^2[<]$ follows from the decidability of $\Sigma_2[<]$, but there is also a more direct effective characterization: A language over finite words is definable in $\mathrm{FO}^2[<]$ if and only if its syntactic monoid is in the variety $\mathbf{DA}$, and the latter property is decidable. The wreath product principle yields $\mathbf{DA} * \mathbf{D}$ as an algebraic characterization of $\mathrm{FO}^2[<, +1]$, but this does not immediately help with decidability. Almeida [1] has shown that $\mathbf{DA} * \mathbf{D} = \mathbf{LDA}$. Now, since $\mathbf{LDA}$ is decidable, membership in $\mathrm{FO}^2[<, +1]$ is decidable. Note that min and max do not yield additional expressive power for $\Delta_2[<]$ and $\mathrm{FO}^2[<]$.

Some of the characterizations and decidability results for the quantifier alternation hierarchy and for $\mathrm{FO}^2[<]$ have been extended to infinite words. Decidability of $\Sigma_1[<]$ and its Boolean closure $\mathbb{B}\Sigma_1[<]$ over infinite words is due to Perrin and Pin [15]; decidability of $\Sigma_2[<]$ over infinite words was shown by Bojańczyk [3]. The fragments $\Delta_2[<]$ and $\mathrm{FO}^2[<]$ do not coincide for infinite words. In particular, decidability of $\mathrm{FO}^2[<]$ does not follow from the respective result for $\Delta_2[<]$. Decidability of $\mathrm{FO}^2[<]$ over infinite words was first shown by Wilke [27].

Over infinite words, using a conjunction of algebraic and topological properties yields further effective characterizations of the fragments $\Sigma_2[<]$ and $\mathrm{FO}^2[<]$, cf. [7]. The key ingredient is the alphabetic topology which is a refinement of the usual Cantor topology. In addition, languages in $\mathrm{FO}^2[<] \cap \Sigma_2[<]$ can be characterized using topological notions; namely, a language L over infinite words is in $\mathrm{FO}^2[<] \cap \Sigma_2[<]$ if and only if L is the interior of a language in $\mathrm{FO}^2[<]$ with respect to the alphabetic topology. By complementation, a language is in $\mathrm{FO}^2[<] \cap \Pi_2[<]$ if and only if it is the topological closure of a language in $\mathrm{FO}^2[<]$. This shows that topology reveals natural properties of first-order fragments over infinite words. In this paper, we continue this line of work.

Outline We combine algebraic and topological properties in order to give effective characterizations of $\Sigma_2[<, +1]$ (Theorem 3.1) and $\mathrm{FO}^2[<, +1]$ (Theorem 4.1) over finite and infinite words. The key ingredient is a generalization of the alphabetic topology which we call the *factor topology*. As a byproduct, we give a new effective characterization of $\Sigma_2[<, +1]$ over finite words (Theorem 3.2), i.e., of the level 3/2 of the dot-depth hierarchy. Dually, we get a characterization of $\Pi_2[<, +1]$ over infinite words (Theorem 3.4). Moreover, we also obtain decidability results for the respective fragments over infinite words (in contrast to finite and infinite words simultaneously; Corollary 3.3 and Corollary 4.2). Concerning the intersection of fragments, we show that L is in $\mathrm{FO}^2[<, +1] \cap \Sigma_2[<, +1]$ if and only if L is the interior of a language in $\mathrm{FO}^2[<, +1]$ with respect to the factor topology (Theorem 6.1) and dually, L is definable in $\mathrm{FO}^2[<,+1] \cap \Pi_2[<,+1]$ if and only if L is the topological closure of a language in $\mathrm{FO}^2[<,+1]$ with respect to the factor topology (Theorem 6.2). Finally, we show that $\Delta_2[<,+1]$ is a strict subclass of $\mathrm{FO}^2[<,+1]$ and that a language L is in $\Delta_2[<,+1]$ if and only if L is in $\mathrm{FO}^2[<,+1]$ and clopen in the factor topology (Theorem 5.1).

Due to lack of space, some proofs are omitted. For complete proofs, we refer to the full version of this paper [11].

2 Preliminaries

Words Throughout, Γ is a finite alphabet and unless stated otherwise u, v, w are finite words, and α, β, γ are finite or infinite words over the alphabet Γ. The set of all finite words is Γ^* and the set of all infinite words is Γ^ω. The empty word is denoted by 1. We write Γ^∞ for the set of all finite and infinite words $\Gamma^* \cup \Gamma^\omega$. As usual, Γ^+ is the set of all non-empty finite words $\Gamma^* \setminus \{1\}$. If L is a subset of a monoid, then L^* is the submonoid generated by L. For $L \subseteq \Gamma^*$ we let $L^\omega = \{u_1 u_2 \cdots \mid u_i \in L \text{ for all } i \geq 1\}$ be the set of infinite products. We also let $L^\infty = L^* \cup L^\omega$. The infinite product of the empty word is empty, i.e., we have $1^\omega = 1$. Thus, $L^\infty = L^\omega$ if and only if $1 \in L$. The *length* of a word $w \in \Gamma^*$ is denoted by $|w|$. We write Γ^k for all words of length k and $\Gamma^{\geq k}$ is the set of finite words of length at least k; similarly, $\Gamma^{<k}$ consist of all words of length less than k. By $\mathrm{alph}_k(\alpha)$ we denote the factors of length k of α, i.e.,

$$\mathrm{alph}_k(\alpha) = \left\{ w \in \Gamma^k \;\middle|\; \alpha = vw\beta \text{ for some } v \in \Gamma^*, \beta \in \Gamma^\infty \right\}.$$

As a special case, we have that $\mathrm{alph}_1(\alpha) = \mathrm{alph}(\alpha)$ is the *alphabet* (also called *content*) of α. We write $\mathrm{im}_k(\alpha)$ for those factors in $\mathrm{alph}_k(\alpha)$ which have infinitely many occurrences in α. The notation $\mathrm{im}_k(\alpha)$ comes from "*im*aginary".

Languages We introduce a non-standard composition $\circ$ for sufficiently long words. Let $k \geq 1$. For $u \in \Gamma^*$ and $\alpha \in \Gamma^\infty$ define $w \circ_k \alpha$ by

$$w \circ_k \alpha = vx\beta \quad \text{if there exists } x \in \Gamma^{k-1} \text{ such that } w = vx \text{ and } \alpha = x\beta.$$

Furthermore $w \circ_k 1 = w$ and $1 \circ_k \alpha = \alpha$. In all other cases $w \circ_k \alpha$ is undefined. Note that $\mathrm{alph}_k(u \circ_k \alpha) = \mathrm{alph}_k(u) \cup \mathrm{alph}_k(\alpha)$, if $u \circ_k \alpha$ is defined. In particular, the operation $\circ_k$ does not introduce new factors of length k. For $A \subseteq \Gamma^k$ we define

$$\begin{aligned}
A^{*_k} &= \{w_1 \circ_k \cdots \circ_k w_n \mid n \geq 0,\ w_i \in A\}, \\
A^{\omega_k} &= \{w_1 \circ_k w_2 \circ_k \cdots \mid w_i \in A\}, \\
A^{\infty_k} &= A^{*_k} \cup A^{\omega_k}, \\
A^{\mathrm{im}_k} &= \{\alpha \in \Gamma^\infty \mid \mathrm{im}_k(\alpha) = A\}.
\end{aligned}$$

If k is clear from the context, then we write $w \circ \alpha$ instead of $w \circ_k \alpha$, we write $A^{\circledast}$ instead of A^{*_k}, we write $A^{\textcircled{$\infty$}}$ instead of A^{∞_k}, and we write $A^{\textcircled{im}}$ instead of A^{im_k}. Note that $\Gamma^* = \emptyset^{\textcircled{im}}$.

A *k-factor monomial* is a language of the form

$$P = A_1^{\circledast} \circ u_1 \circ \cdots \circ A_s^{\circledast} \circ u_s \circ A_{s+1}^{\textcircled{∞}}$$

for $u_i \in \Gamma^{\geq k}$ and $A_i \subseteq \Gamma^k$. The *degree* of P is the length of the word $u_1 \cdots u_s$. A *k-factor polynomial* is a finite union of k-factor monomials and of words of length less than k. A language L is a factor polynomial (resp. monomial) if there is a number k such that L is a k-factor polynomial (resp. monomial).

Fragments of First-order Logic We think of words as labeled linear orders, and we write $x < y$, if position x comes before position y. Similarly, $x = y + 1$ means that x is the successor of y. A position x of a word α is an *a-position*, if the label of x in α is the letter a.

We denote by FO the first-order logic over words. Atomic formulas in FO are $\top$ (for *true*), unary predicates $\lambda(x) = a$ for $a \in \Gamma$, and binary predicates $x < y$ and $x = y{+}1$ for variables x and y. Variables range over positions in $\mathbb{N}$ and $\lambda(x) = a$ means that x is an a-position. Formulas may be composed using Boolean connectives as well as existential quantification $\exists x \colon \varphi$ and universal quantification $\forall x \colon \varphi$ for $\varphi \in \mathrm{FO}$. The semantics is as usual. A *sentence* in FO is a formula without free variables. Let $\varphi \in \mathrm{FO}$ be a sentence. We write $\alpha \models \varphi$ if α models φ. The *language defined by* φ is $L(\varphi) = \{\alpha \in \Gamma^\infty \mid \alpha \models \varphi\}$.

The fragment $\Sigma_n[\mathcal{C}]$ of FO for $\mathcal{C} \subseteq \{<, +1\}$ consists of all sentences in prenex normal form with n blocks of quantifiers starting with a block of existential quantifiers. In addition, only binary predicates in $\mathcal{C}$ are allowed. The fragment $\Pi_n[\mathcal{C}]$ consists of negations of formulas in $\Sigma_n[\mathcal{C}]$. We frequently identify first-order fragments with the classes of languages they define. For example, $\Delta_n[\mathcal{C}] = \Sigma_n[\mathcal{C}] \cap \Pi_n[\mathcal{C}]$ is the class of all languages which are definable in both $\Sigma_n[\mathcal{C}]$ and $\Pi_n[\mathcal{C}]$. Another important fragment is $\mathrm{FO}^2[\mathcal{C}]$. It consists of all sentences using (and reusing) only two different names for the variables, say x and y, and where only binary predicates from $\mathcal{C}$ are allowed. Let $\mathcal{F}$ be a fragment of first-order logic. We say that L is *$\mathcal{F}$-definable over some subset* $K \subseteq \Gamma^\infty$, if there exists some formula $\varphi \in \mathcal{F}$ with $L = \{\alpha \in K \mid \alpha \models \varphi\}$. We frequently use this notion for either $K = \Gamma^*$ or $K = \Gamma^\omega$.

Finite Monoids We repeat some basic notions and properties concerning finite monoids. For further details we refer to standard textbooks such as [16]. Let M be a finite monoid. For every such monoid there exists a number $n \geq 1$ such that $a^n = a^{2n}$ for all $a \in M$, i.e., a^n is the unique idempotent power of a. The set of all idempotents of M is denoted by $E(M)$. An important tool in the study of finite monoids are *Green's relations.* At this point, we only introduce their ordered versions. We have $a \leq_{\mathcal{R}} b$ if and only if $aM \subseteq bM$, we have $a \leq_{\mathcal{L}} b$ if and only if $Ma \subseteq Mb$, and we have $a \leq_{\mathcal{J}} b$ if and only if $MaM \subseteq MbM$.

An *ordered monoid* M is equipped with a partial order $\leq$ which is compatible with multiplication, i.e., $a \leq b$ and $c \leq d$ implies $ac \leq bd$. We can always assume that M is ordered, since equality is a compatible partial order.

The theory of first-order fragments over finite non-empty words is presented more concisely in the context of semigroups instead of monoids. In this paper however, we want to incorporate finite and infinite words in a uniform model, and our approach is heavily based on allowing words to be empty. In order to state "semigroup conditions" for monoids, we have to use surjective homomorphisms $h : \Gamma^* \to M$ instead of monoids M only.

Let $h : \Gamma^* \to M$ be a surjective homomorphism and let $e \in M$ be an idempotent. The set P_e consists of all products of the form $x_0 f_1 \cdots x_{m-1} f_m x_m$ with elements $x_0, \ldots, x_m \in M$ and idempotents $f_1, \ldots, f_m \in h(\Gamma^+) \subseteq M$ satisfying the following three conditions

$$\begin{aligned} &e \leq_{\mathcal{R}} x_0 f_1, \\ &e \leq_{\mathcal{J}} f_i x_i f_{i+1} \qquad \text{for all } 1 \leq i \leq m-1, \\ &e \leq_{\mathcal{L}} f_m x_m. \end{aligned}$$

If $e \notin h(\Gamma^+)$, then we set $P_e = \{1\}$. Note that in this case we necessarily have $e = 1$ in M. The notation P_e is for *paths in* e. An idempotent e is said to be *locally path-top* with respect to h if $eP_e e \leq e$. Symmetrically, it is *locally path-bottom* with respect to h if $eP_e e \geq e$. If the underlying homomorphism is clear from the context, we omit the reference to it. The homomorphism h is *locally path-top* (resp. *locally path-bottom*) if all idempotents in M are locally path-top (resp. locally path-bottom).

▶ **Lemma 2.1.** *Let $h : \Gamma^* \to M$ be a surjective homomorphism onto a finite monoid M. It is decidable whether M is locally path-top.*

Proof. We give an algorithm computing P_e for a given idempotent e. We define a composition on triples $T = E(M) \times M \times E(M)$ by $(f_1, x_1, f_2)(f_3, x_2, f_4) = (f_1, x_1 f_2 x_2, f_4)$ if $f_2 = f_3$; otherwise the composition is undefined. Compute the fixed point P of the equation $P = P \cup P T_e$ with $T_e = \{(f_1, x_1, f_2) \in T \mid f_1, f_2 \in h(\Gamma^+),\ e \leq_{\mathcal{J}} f_1 x_1 f_2\}$ and initial value $P = T_e$. This requires at most $|M|^3$ iterations. Then P_e is the set of all $x_0 f_1 x f_2 x_2$ where $(f_1, x, f_2) \in P$, $e \leq_{\mathcal{R}} x_0 f_1$ and $e \leq_{\mathcal{L}} f_2 x_2$. ◀

Let $h : \Gamma^* \to M$ be a surjective homomorphism and let $n \in \mathbb{N}$ such that a^n is idempotent for all $a \in M$. The homomorphism $h : \Gamma^* \to M$ is in **LDA** if

$$(eaebe)^n \, eae \, (eaebe)^n = (eaebe)^n$$

for all idempotents $e \in h(\Gamma^+)$ and for all $a, b \in M$. If the reference to the homomorphism is clear from the context, then we say "$M \in \mathcal{P}$" for some property $\mathcal{P}$ meaning that "$h \in \mathcal{P}$".

Recognizability A language $L \subseteq \Gamma^\infty$ is *regular* if it is recognized be some extended Büchi automaton, see e.g. [6], or equivalently, if it is definable in monadic second order logic [26]. Below, we present a more algebraic framework for recognition of $L \subseteq \Gamma^\infty$. The *syntactic preorder* $\leq_L$ over Γ^* is defined as follows. We let $s \leq_L t$ if for all $u, v, w \in \Gamma^*$ we have the following two implications:

$$utvw^\omega \in L \ \Rightarrow\ usvw^\omega \in L \qquad \text{and} \qquad u(tv)^\omega \in L \ \Rightarrow\ u(sv)^\omega \in L.$$

Remember that $1^\omega = 1$. Two words s, $t \in \Gamma^*$ are syntactically equivalent, written as $s \equiv_L t$, if both $s \leq_L t$ and $t \leq_L s$. This is a congruence and the congruence classes $[s]_L = \{t \in \Gamma^* \mid s \equiv_L t\}$ form the *syntactic monoid* $\mathrm{Synt}(L)$ of L. The preorder $\leq_L$ on words induces a partial order $\leq_L$ on congruence classes, and $(\mathrm{Synt}(L), \leq_L)$ becomes an ordered monoid. It is a well-known classical result that the syntactic monoid of a regular language $L \subseteq \Gamma^\infty$ is finite, see e.g. [15, 26]. Moreover, in this case L can be written as a finite union of languages of the form $[s]_L \, [t]_L^\omega$ with s, $t \in \Gamma^*$ and $st \equiv_L s$ and $t^2 \equiv_L t$.

Now, let $h : \Gamma^* \to M$ be any surjective homomorphism onto a finite ordered monoid M and let $L \subseteq \Gamma^\infty$. If the reference to h is clear from the context, then we denote by $[s]$ the set of finite words $h^{-1}(s)$ for $s \in M$. The following notations are used:

- $(s, e) \in M \times M$ is a *linked pair*, if $se = s$ and $e^2 = e$.
- h *weakly recognizes* L, if
$$L = \bigcup \{[s][e]^\omega \mid (s, e) \text{ is a linked pair and } [s][e]^\omega \subseteq L\}\,.$$
- h *strongly recognizes* L (or simply *recognizes* L), if
$$L = \bigcup \{[s][e]^\omega \mid (s, e) \text{ is a linked pair and } [s][e]^\omega \cap L \neq \emptyset\}\,.$$
- L is *downward closed (on finite prefixes)* for h, if $[s][e]^\omega \subseteq L$ implies $[t][e]^\omega \subseteq L$ for all $s, t, e \in M$ where $t \leq s$.

Using Ramsey's Theorem, one can show that for every word $\alpha \in \Gamma^\infty$ there exists a linked pair (s, e) such that $\alpha \in [s][e]^\omega$. On the other hand, two different languages of the form $[s][e]^\omega$ are not necessarily disjoint. Therefore, if L is weakly recognized by h, then there could exist some linked pair (s, e) such that $[s][e]^\omega$ and L are incomparable. If L is strongly recognized by h, then for every linked pair we have either $[s][e]^\omega \subseteq L$ or $[s][e]^\omega \cap L = \emptyset$. In particular, whenever L is strongly recognized by h, then $\Gamma^\infty \setminus L$ is also strongly recognized by h. Every regular language L is strongly recognized by its syntactic homomorphism $h_L : \Gamma^* \to \mathrm{Synt}(L);\ s \mapsto [s]_L$. Moreover, L is downward closed for h_L.

2.1 The factor topology

Topological properties play a crucial role in this paper. Very often a combination of algebraic and topological properties yields a decidable characterization of the fragments. Moreover, topology can be used to describe the relation between the fragments. This section introduces the topology matching the fragments $\Sigma_2[<, +1]$ and $\Pi_2[<, +1]$.

We define the *k-factor topology* by its basis. All sets of the form $u \circ A^{\otimes}$ for $u \in \Gamma^*$ and $A \subseteq \Gamma^k$ are open. Therefore, singleton sets $\{u\}$ for $u \in \Gamma^*$ are open in the k-factor topology since $\{u\} = u \circ \emptyset^{\otimes}$. A language is said to be *factor open* (resp. *factor closed*) if there is a natural number k such that L is open (resp. closed) in the k-factor topology.

▶ **Proposition 2.2.** *Let $L \subseteq \Gamma^\infty$ be a regular language. Then L is factor open if and only if L is open in the $(2\,|\mathrm{Synt}(L)|)$-factor topology.*

▶ **Proposition 2.3.** *It is decidable whether a regular language $L \subseteq \Gamma^\infty$ is factor open.*

3 The first-order fragment Σ_2

One of our main results is a decidable characterization of the fragment $\Sigma_2[<, +1]$ over finite and infinite words. It is a combination of a decidable algebraic and a decidable topological property. For finite words only, this yields a new decidable algebraic characterization for dot-depth 3/2, which in turn coincides with $\Sigma_2[<, +1]$ over finite words [25].

▶ **Theorem 3.1.** *Let $L \subseteq \Gamma^\infty$ be a regular language. The following are equivalent:*

1. *L is $\Sigma_2[<, +1]$-definable.*
2. *L is a factor polynomial.*
3. *L is factor open and there exists a surjective locally path-top homomorphism $h : \Gamma^* \to M$ which weakly recognizes L such that L is downward closed for h.*
4. *L is factor open and $\mathrm{Synt}(L)$ is locally path-top.*

Next, we give a counterpart of the preceding theorem for finite words, which in turn yields a new decidable characterization of dot-depth 3/2. The first decidable characterization was discovered by Glaßer and Schmitz [9, 10]. It is based on so-called forbidden patterns. Later, a decidable algebraic characterization was given by Pin and Weil [19].

▶ **Theorem 3.2.** *Let $L \subseteq \Gamma^*$ be a language. The following are equivalent over finite words:*

1. *L is $\Sigma_2[<, +1]$-definable over finite words.*
2. *L is a factor polynomial.*
3. *$\mathrm{Synt}(L)$ is finite and locally path-top.*

Proof. The language Γ^* of finite words is definable in $\Sigma_2[<]$ by stating that there is a position such that all other positions are smaller. Hence, if $L = \{w \in \Gamma^* \mid w \models \varphi\}$ for some

$\varphi \in \Sigma_2[<,+1]$, then there also exists some $\varphi' \in \Sigma_2[<,+1]$ such that $L = \{\alpha \in \Gamma^\infty \mid \alpha \models \varphi'\}$. Using Theorem 3.1, this shows "1 ⇒ 2". Trivially, "2 ⇒ 3" follows from the same theorem. Finally, "3 ⇒ 1" uses the fact that every language over finite words is factor open. ◀

The equivalence of (1) and (2) in Theorem 3.2 was also shown by Glaßer and Schmitz using different techniques and with another formalism for defining factor polynomials [10]. As a corollary of Theorem 3.1 and Theorem 3.2 we obtain the following decidability results.

▶ **Corollary 3.3.** *Let L be a regular language.*

1. *For $L \subseteq \Gamma^\infty$ it is decidable, whether L is $\Sigma_2[<,+1]$-definable.*
2. *For $L \subseteq \Gamma^*$ it is decidable, whether L is $\Sigma_2[<,+1]$-definable over finite words.*
3. *For $L \subseteq \Gamma^\omega$ it is decidable, whether L is $\Sigma_2[<,+1]$-definable over infinite words.*

Proof. For "1" we note that the syntactic monoid is effectively computable. Therefore, Theorem 3.1 (4) can be verified effectively by Lemma 2.1 and Proposition 2.3. Similarly, "2" follows from the decidability of Theorem 3.2 (3). The set Γ^* is definable in $\Sigma_2[<,+1]$ over Γ^∞. Hence, $L \subseteq \Gamma^\omega$ is $\Sigma_2[<,+1]$-definable over Γ^ω if and only if $L \cup \Gamma^*$ is $\Sigma_2[<,+1]$-definable over Γ^∞, and the latter condition is decidable by "1". Therefore, assertion "3" holds. ◀

By duality, the properties of $\Sigma_2[<,+1]$ in Theorem 3.1 yield a decidable characterization of $\Pi_2[<,+1]$, which we state here for completeness.

▶ **Theorem 3.4.** *Let $L \subseteq \Gamma^\infty$ be a regular language. The following are equivalent:*

1. *L is $\Pi_2[<,+1]$-definable.*
2. *L is factor closed and* $\mathrm{Synt}(L)$ *is locally path-bottom.*

4 First-order logic with two variables

In this section, we consider two-variable first-order logic with order $<$ and successor $+1$ over finite and infinite words. The fragment $\mathrm{FO}^2[<,+1]$ admits a temporal logic counterpart having the same expressive power [8]. It is based on unary modalities only. Wilke [27] has shown that membership is decidable for $\mathrm{FO}^2[<,+1]$. We complement these results by giving a simple algebraic characterization of this fragment. An important concept in our proof is a refinement of the factor topology. A set of the form $A^{\textcircled{im}}$ is definable in $\mathrm{FO}^2[<,+1]$ but it is neither open nor closed in the factor topology. This observation leads to the *strict k-factor topology*. A basis of this topology is given by all sets of the form $u \circ A^{\textcircled{$\infty$}} \cap A^{\textcircled{im}}$ for $u \in \Gamma^*$ and $A \subseteq \Gamma^k$. We do not use this topology outside this section.

▶ **Theorem 4.1.** *Let $L \subseteq \Gamma^\infty$ be a regular language. The following are equivalent:*

1. *L is $\mathrm{FO}^2[<,+1]$-definable.*
2. *L is weakly recognized by some homomorphism $h : \Gamma^* \to M \in \mathbf{LDA}$ and closed in the strict $(2\,|M|)$-factor topology.*
3. $\mathrm{Synt}(L) \in \mathbf{LDA}$.

The syntactic monoid of a regular language is effectively computable. Hence, one can verify whether property (3) in Theorem 4.1 holds. Since both Γ^* and Γ^ω are $\mathrm{FO}^2[<,+1]$-definable over Γ^∞, this gives us the following corollary.

▶ **Corollary 4.2.** *Let L be a regular language.*

1. *For $L \subseteq \Gamma^\infty$ it is decidable, whether L is $\mathrm{FO}^2[<,+1]$-definable.*
2. *For $L \subseteq \Gamma^*$ it is decidable, whether L is $\mathrm{FO}^2[<,+1]$-definable over finite words.*
3. *For $L \subseteq \Gamma^\omega$ it is decidable, whether L is $\mathrm{FO}^2[<,+1]$-definable over infinite words.*

The following proposition relates monoids in **LDA** with monoids which are simultaneously locally path-top and locally path-bottom. It is a useful tool in the proof of Theorem 4.1. Moreover, it immediately follows that $\Delta_2[<,+1]$ is a subclass of $\mathrm{FO}^2[<,+1]$. We will further explore the relation between these two fragments in the next section.

▶ **Proposition 4.3.** *Let M be finite and let $h : \Gamma^* \to M$ be a surjective homomorphism. The following are equivalent:*
1. $h : \Gamma^* \to M \in \mathbf{LDA}$.
2. $eP_e e = e$ *for all idempotents e of M.*

▶ **Example 4.4.** Let $\Gamma = \{a, b, c\}$. Consider the language $L_1 = \Gamma^* ab^* a\Gamma^\infty$ consisting of all words such that there are two a's that only contain b's in between. It is easy to see that L_1 is $\Sigma_2[<]$-definable. Next, we will show that L_1 is not $\mathrm{FO}^2[<,+1]$-definable. Choose $n \in \mathbb{N}$ such that s^n is idempotent for every $s \in \mathrm{Synt}(L_1)$. Then $(b^n ab^n cb^n)^n \notin L_1$ whereas $(b^n ab^n cb^n)^n b^n ab^n (b^n ab^n cb^n)^n \in L_1$. This shows that $\mathrm{Synt}(L_1)$ is not in **LDA**. By Theorem 4.1 we conclude that L_1 is not $\mathrm{FO}^2[<,+1]$-definable. Similarly, $L_2 = \Gamma^\infty \setminus L_1$ is definable in $\Pi_2[<]$ but not in $\mathrm{FO}^2[<,+1]$. ◁

5 The first-order fragment Δ_2

Over finite words, the first-order fragments $\mathrm{FO}^2[<,+1]$ and $\Delta_2[<,+1]$ have the same expressive power [14, 24]. This is not true for infinite words. Here, it turns out that $\Delta_2[<,+1]$ is a strict subclass of $\mathrm{FO}^2[<,+1]$ and that the $\Delta_2[<,+1]$-languages are exactly the clopen languages in $\mathrm{FO}^2[<,+1]$.

▶ **Theorem 5.1.** *Let $L \subseteq \Gamma^\infty$ be a language. The following are equivalent:*
1. *L is $\Delta_2[<,+1]$-definable.*
2. *L is $\mathrm{FO}^2[<,+1]$-definable and clopen in the factor topology.*

A consequence of Theorem 5.1 is that $\Delta_2[<,+1]$ is a strict subclass of $\mathrm{FO}^2[<,+1]$. In fact, it is a strict subclass of the intersection for the fragments $\mathrm{FO}^2[<,+1]$ and $\Sigma_2[<,+1]$.

▶ **Corollary 5.2.** *Over Γ^∞, the fragment $\Delta_2[<,+1]$ is a strict subclass of the fragment $\mathrm{FO}^2[<,+1] \cap \Sigma_2[<,+1]$ and also of the fragment $\mathrm{FO}^2[<,+1] \cap \Pi_2[<,+1]$.*

Proof. The set of non-empty finite words Γ^+ is defined by the sentence

$$\exists x\, \forall y\colon y \le x$$

in $\mathrm{FO}^2[<] \cap \Sigma_2[<]$. We have to show that Γ^+ is not definable in $\Pi_2[<,+1]$. By Theorem 3.4 it suffices to show that Γ^+ is not factor closed. Let $a \in \Gamma$, and consider the word $\alpha = a^\omega \notin \Gamma^+$. Every factor open set containing α also contains some finite word $a^m \in \Gamma^+$. Hence, the complement of Γ^+ is not factor open, and therefore, Γ^+ is not factor closed. By complementation, we see that Γ^ω is definable in $\mathrm{FO}^2[<] \cap \Pi_2[<]$ but not in $\Delta_2[<,+1]$. ◀

▶ **Example 5.3.** We consider another language which is definable in $\mathrm{FO}^2[<] \cap \Sigma_2[<]$ but not in $\Delta_2[<,+1]$. Let $\Gamma = \{a, b\}$ and $L_3 = \Gamma^* ab^\infty$. The language L_3 is $\mathrm{FO}^2[<] \cap \Sigma_2[<]$-definable:

$$\exists x\, \forall y\colon \lambda(x) = a \;\wedge\; \big(\lambda(y) = a \;\Rightarrow\; y \le x\big).$$

In order to show that L_3 is not definable in $\Pi_2[<,+1]$, it suffices to show that L_3 is not factor closed (Theorem 3.4). Let $k \in \mathbb{N}$. Every open set containing the word $(b^k a)^\omega \notin L_3$

also contains some word $(b^k a)^m b^\omega \in L_3$. Hence, the complement of L_3 is not k-factor open, and therefore, there is no k such that L_3 is closed in the k-factor topology.

The same reasoning also works over Γ^ω, since the language of all infinite words is definable in $\Pi_2[<,+1]$. Hence, $L_3' = \Gamma^* ab^\omega$ is definable in $\Sigma_2[<]$ over infinite words and in $\mathrm{FO}^2[<]$ but not in $\Delta_2[<,+1]$ over infinite words. The language L_3' is the standard example of a language which cannot be recognized by a deterministic Büchi automaton [26, Example 4.2]. In particular, none of the fragments $\mathrm{FO}^2[<,+1]$ or $\Sigma_2[<,+1]$ contains only deterministic languages. ◁

▶ **Example 5.4.** Let $\Gamma = \{a, b, c\}$ and consider the language $L_4 = (\Gamma^2 \setminus \{bb\})^\circledast \circ aa \circ (\Gamma^2)^\circledast$ consisting of all words such that there is no factor bb before the first factor aa. The language L_4 is defined by the $\Sigma_2[<,+1]$-sentence

$$\exists x \forall y < x\colon \lambda(x) = aa \wedge \lambda(y) \neq bb.$$

Here, $\lambda(x) = w$ is a shortcut saying that a factor w starts at position x. A word α is in L_4 if and only if aa is a factor of α and for every factor bb there is a factor aa to the left. These properties are $\Pi_2[<,+1]$-definable and hence $L_4 \in \Delta_2[<,+1]$. The language L_4 is not definable in any of the fragments $\mathrm{FO}^2[<]$, $\Sigma_2[<]$, or $\Pi_2[<]$ without successor, since its syntactic monoid is neither locally top nor locally bottom, cf. [7]. The language $L_4 \cap \Gamma^*$ has been used as an example of a language not definable in the Boolean closure of $\Sigma_2[<]$ over finite words by Almeida and Klíma [2, Proposition 6.1] as well as by Lodaya, Pandya, and Shah [14, Theorem 4]. The Boolean closure of $\Sigma_2[<]$ over finite words coincides with the second level of the Straubing-Thérien hierarchy, cf. [18, 25]. ◁

6 The first-order fragments $\mathrm{FO}^2 \cap \Sigma_2$ and $\mathrm{FO}^2 \cap \Pi_2$

In this section, we show that topological concepts can not only be used as an ingredient for characterizing first-order fragments, but also for describing some relations between fragments. More precisely, we relate languages definable in both $\Sigma_2[<,+1]$ and $\mathrm{FO}^2[<,+1]$ with the interiors of $\mathrm{FO}^2[<,+1]$-languages with respect to the factor topology. Dually, the languages in the fragment $\mathrm{FO}^2[<,+1] \cap \Pi_2[<,+1]$ are precisely the topological closures of $\mathrm{FO}^2[<,+1]$-languages. Remember that for a language L, its *closure* $\overline{L}$ is the intersection of all closed sets containing L. It can be "computed" as

$$\overline{L} = \{\alpha \in \Gamma^\infty \mid \forall U \subseteq \Gamma^\infty \text{ open with } \alpha \in U\colon U \cap L \neq \emptyset\}.$$

The *interior* of L is the union of all open sets contained in L. The interior of a language is the complement of the closure of its complement.

▶ **Theorem 6.1.** *Let $L \subseteq \Gamma^\infty$ be a regular language. The following are equivalent:*

1. $L \in \mathrm{FO}^2[<,+1] \cap \Sigma_2[<,+1]$.
2. $L \in \mathrm{FO}^2[<,+1]$ *and L is open in the factor topology.*
3. *L is the factor interior of some* $\mathrm{FO}^2[<,+1]$*-definable language.*

The equivalence of (1) and (2) is an immediate consequence of Theorems 3.1 and 4.1. The surprising property is (3); for example, it is not obvious that the factor interior of an $\mathrm{FO}^2[<,+1]$-definable language is again in $\mathrm{FO}^2[<,+1]$. The following theorem is an immediate consequence of Theorem 6.1, obtained by complementation. In fact, the actual proof is slightly easier the other way round — proving Theorem 6.2 and then obtaining Theorem 6.1 by complementation — since the closure of a language is easier to "compute".

▶ **Theorem 6.2.** *Let $L \subseteq \Gamma^\infty$ be a regular language. The following are equivalent:*

1. $L \in \mathrm{FO}^2[<,+1] \cap \Pi_2[<,+1]$.
2. $L \in \mathrm{FO}^2[<,+1]$ *and L is closed in the factor topology.*
3. *L is the factor closure of some $\mathrm{FO}^2[<,+1]$-definable language.*

The language L_3 from Example 5.3 is definable in $\mathrm{FO}^2[<,+1] \cap \Sigma_2[<,+1]$. For any k, the k-factor closure of L_3 is Γ^∞. Hence, one could conjecture that the factor closure of every language in $\mathrm{FO}^2[<,+1] \cap \Sigma_2[<,+1]$ is again in $\mathrm{FO}^2[<,+1] \cap \Sigma_2[<,+1]$ and therefore in $\Delta_2[<,+1]$. Among other things, the following example shows that this is not the case.

▶ **Example 6.3.** Let $\Gamma = \{a,b,c\}$. We consider the factor closure of the language L_5 defined by the $\mathrm{FO}^2[<,+1]$-sentence

$$\exists x\colon \lambda(x) = aba \wedge \big(\exists y > x\colon \lambda(y) = aba\big) \wedge \big(\neg\exists y > x\colon \lambda(y) = bab\big)$$

with $\lambda(x) = w$ being a macro for "an occurrence of the factor $w \in \Gamma^+$ starts at position x". The language L_5 contains all words of the form $u \cdot aba \cdot v \cdot aba \cdot \beta$ with $u, v \in \Gamma^*$, $\beta \in \Gamma^\infty$, and $bab \notin \mathrm{alph}_3(aba \cdot v \cdot aba \cdot \beta)$.

The $(k+1)$-factor closure of a language is always contained in its k-factor closure. For L_5, we show that this inclusion is strict for $k \in \{1,2\}$. The word $(ab)^\omega$ is in the 1-factor closure of L_5, but not in its 2-factor closure. The word $(abacbabc)^\omega$ is in the 2-factor closure of L_5, but not in its 3-factor closure. The 1-factor closure of L_5 is $L_5 \cup \{\alpha \in \Gamma^\omega \mid a, b \in \mathrm{im}_1(\alpha)\}$. The 2-factor closure of L_5 consists of L_5 and all words α with either $\{ab, ba, aa\} \subseteq \mathrm{im}_2(\alpha)$ or $\{ab, ba, ac, ca\} \subseteq \mathrm{im}_2(\alpha)$. The 3-factor closure of L_5 is similar, but it requires more case distinctions.

There is no k such that L_5 is k-factor closed. By Theorem 3.4, we see that L_5 is not $\Pi_2[<,+1]$-definable. On the other hand, Theorem 6.2 says that every k-factor closure of L_5 is $\Pi_2[<,+1]$-definable. Moreover, almost the same sentence as above yields $\Sigma_2[<,+1]$-definability of L_5. Since the 1-factor closure of L_5 is not factor open, the fragment $\mathrm{FO}^2[<,+1] \cap \Sigma_2[<,+1]$ is not closed under taking factor closures, whereas $\mathrm{FO}^2[<,+1]$ has this closure property by Theorem 6.2. ◁

7 Summary

We considered fragments of first-order logic over finite and infinite words. As binary predicates we allow order comparison $x < y$ and the successor predicate $x = y + 1$. Figure 1 depicts the relation between the fragments $\Sigma_2[<,+1]$, $\Pi_2[<,+1]$, and $\mathrm{FO}^2[<,+1]$. Moreover, the languages L_1, L_2, L_3, L_4, and L_5 from Examples 4.4, 5.3, 5.4, and 6.3 are included. For the other languages, we fix $\Gamma = \{a,b,c\}$ and $\emptyset \neq A \subsetneq \Gamma$.

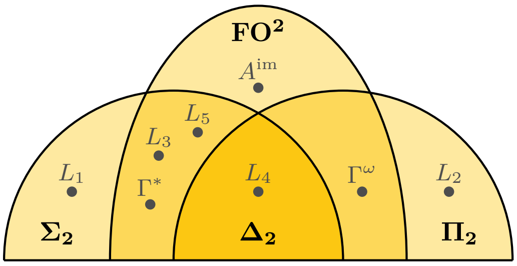


Figure 1 The fragments $\Sigma_2[<,+1]$, $\Pi_2[<,+1]$, and $\mathrm{FO}^2[<,+1]$ over Γ^∞.

The central notion for presenting our results is a partially defined composition $u \circ_k v = u'xv'$ where $u = u'x$, $v = xv'$, and $|x| = k - 1$. Using this composition, one can show that the languages definable in $\Sigma_2[<, +1]$ is exactly the class of factor polynomials. Moreover, the composition $\circ_k$ leads to the k-factor topology, which we use in further characterizations of the successor fragments. A set is *factor open* if there exists some number k such that L is k-factor open. For every regular language L, Proposition 2.2 gives a bound k such that L is factor open if and only if L is k-factor open. Then, in Proposition 2.3, we essentially show that for a given number k it is decidable whether a regular language L is k-factor open. Altogether, in order to check whether L is factor open, we can check whether L is k-factor open, with k being the bound given by Proposition 2.2. Hence, the topological properties, which we use in the characterizations of the fragments, are decidable. Together with the decidable algebraic properties, this gives a decision procedure for deciding whether a given regular language $L \subseteq \Gamma^\infty$ or $L \subseteq \Gamma^\omega$ is definable in one of the fragments under consideration. In Table 1 we summarize our main results. All fragments are using binary predicates $[<, +1]$. The first decidable characterization of $\mathrm{FO}^2[<, +1]$ is due to Wilke [27]. Decidability for $\Sigma_2[<, +1]$ over infinite words is new (Corollary 3.3).

Logic	Algebra	+	Topology	Languages	
Σ_2	$eP_e e \leq e$	+	factor open	factor polynomials	Thm. 3.1
Π_2	$eP_e e \geq e$	+	factor closed		Thm. 3.4
FO^2	**LDA**				Thm. 4.1
	weak **LDA**	+	strictly factor closed		
Δ_2	**LDA**	+	factor clopen		Thm. 5.1
$\mathrm{FO}^2 \cap \Sigma_2$	**LDA**	+	factor open	factor interior of FO^2	Thm. 6.1
$\mathrm{FO}^2 \cap \Pi_2$	**LDA**	+	factor closed	factor closure of FO^2	Thm. 6.2

Table 1 Main characterizations of some first-order fragments

Open problems The fragment $\Sigma_2[<, +1]$ has a language description in terms of factor polynomials. Without the successor predicate similar characterizations in terms of so-called unambiguous polynomials exist for the fragments $\mathrm{FO}^2[<]$, for $\mathrm{FO}^2[<] \cap \Sigma_2[<]$, and for $\Delta_2[<]$, cf. [7]. It is open whether these fragments admit similar characterizations if we allow the successor predicate.

Moreover, for the fragment $\Delta_2[<, +1]$ we only have an implicit decidable characterization based on the decidability of $\Sigma_2[<, +1]$ and $\Pi_2[<, +1]$ (or alternatively, based on the decidability of $\mathrm{FO}^2[<, +1]$ and being clopen). A more direct characterization of this fragment remains open. For $\Delta_2[<]$ without successor, such a characterization shows that all languages in $\Delta_2[<]$ over infinite words are recognizable by deterministic Büchi automata.

Another important fragment is $\mathbb{B}\Sigma_1$, the Boolean closure of Σ_1. A result of Knast [13] shows that, over finite words, it is decidable whether a regular language is definable in $\mathbb{B}\Sigma_1[<, +1, \min, \max]$, which over finite words corresponds to the first level of the dot-depth hierarchy. A similar result over infinite words is still missing.

Acknowledgments We would like to thank the anonymous referees for their comments which helped to improve the presentation of this paper.

References

1 J. Almeida. A syntactical proof of locality of **DA**. *Int. J. Algebra Comput.*, 6(2):165–177, 1996.

2 J. Almeida and O. Klíma. New decidable upper bound of the second level in the Straubing-Thérien concatenation hierarchy of star-free languages. *Discrete Math. Theor. Comput. Sci.*, 12(4):41–58, 2010.

3 M. Bojańczyk. The common fragment of ACTL and LTL. In *FoSSaCS 2008, Proceedings*, vol. 4962 of *LNCS*, pp. 172–185. Springer, 2008.

4 J. A. Brzozowski and R. Knast. The dot-depth hierarchy of star-free languages is infinite. *J. Comput. Syst. Sci.*, 16(1):37–55, 1978.

5 R. S. Cohen and J. A. Brzozowski. Dot-depth of star-free events. *J. Comput. Syst. Sci.*, 5(1):1–16, 1971.

6 V. Diekert and P. Gastin. First-order definable languages. In *Logic and Automata: History and Perspectives*, Texts in Logic and Games, pp. 261–306. Amsterdam University Press, 2008.

7 V. Diekert and M. Kufleitner. Fragments of first-order logic over infinite words. To appear in *Theory Comput. Syst.*, Special issue STACS 2009.

8 K. Etessami, M. Y. Vardi, and Th. Wilke. First-order logic with two variables and unary temporal logic. *Inf. Comput.*, 179(2):279–295, 2002.

9 Ch. Glaßer and H. Schmitz. Languages of dot-depth 3/2. In *STACS 2000, Proceedings*, vol. 1770 of *LNCS*, pp. 555–566. Springer, 2000.

10 Ch. Glaßer and H. Schmitz. Languages of dot-depth 3/2. *Theory Comput. Syst.*, 42(2):256–286, 2008.

11 J. Kallas, M. Kufleitner, and A. Lauser. First-order fragments with successor over infinite words. Tech. Rep. no. 2010/08, Computer Science, University of Stuttgart (Germany), 2010.

12 J. A. W. Kamp. *Tense Logic and the Theory of Linear Order.* PhD thesis, University of California, Los Angeles (California), 1968.

13 R. Knast. A semigroup characterization of dot-depth one languages. *RAIRO, Inform. Théor. Appl.*, 17(4):321–330, 1983.

14 K. Lodaya, P. K. Pandya, and S. S. Shah. Around dot depth two. In *DLT 2010, Proceedings*, vol. 6224 of *LNCS*, pp. 305–316. Springer, 2010.

15 D. Perrin and J.-É. Pin. *Infinite words*, vol. 141 of *Pure and Appl. Mathematics*. Elsevier, 2004.

16 J.-É. Pin. *Varieties of Formal Languages.* North Oxford Academic, 1986.

17 J.-É. Pin. Syntactic semigroups. In *Handbook of Formal Languages, Vol. 1*, pp. 679–746. Springer, 1997.

18 J.-É. Pin and P. Weil. Polynomial closure and unambiguous product. *Theory Comput. Syst.*, 30(4):383–422, 1997.

19 J.-É. Pin and P. Weil. The wreath product principle for ordered semigroups. *Commun. Algebra*, 30(12):5677–5713, 2002.

20 H. Straubing. A generalization of the Schützenberger product of finite monoids. *Theor. Comput. Sci.*, 13:137–150, 1981.

21 H. Straubing. Finite semigroup varieties of the form $\mathbf{V}*\mathbf{D}$. *J. Pure Appl. Algebra*, 36(1):53–94, 1985.

22 P. Tesson and D. Thérien. Diamonds are Forever: The Variety DA. In *Semigroups, Algorithms, Automata and Languages, 2001*, pp. 475–500. World Scientific, 2002.

23 D. Thérien. Classification of finite monoids: The language approach. *Theor. Comput. Sci.*, 14(2):195–208, 1981.

24 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.

25 W. Thomas. Classifying regular events in symbolic logic. *J. Comput. Syst. Sci.*, 25:360–376, 1982.

26 W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science*, pp. 133–191. Elsevier, 1990.

27 Th. Wilke. *Classifying Discrete Temporal Properties.* Habilitationsschrift, Universität Kiel, April 1998.

The model checking problem for propositional intuitionistic logic with one variable is AC^1-complete

Martin Mundhenk and Felix Weiß

Institut für Informatik, Universität Jena
Jena, Germany
`{martin.mundhenk,felix.weiss}@uni-jena.de`

Abstract

We investigate the complexity of the model checking problem for propositional intuitionistic logic. We show that the model checking problem for intuitionistic logic with one variable is complete for logspace-uniform AC^1, and for intuitionistic logic with two variables it is P-complete. For superintuitionistic logics with one variable, we obtain NC^1-completeness for the model checking problem and for the tautology problem.

1998 ACM Subject Classification F.2 Analysis of algorithms and problem complexity, F.4 Mathematical logic and formal languages

Keywords and phrases complexity, intuitionistic logic, model checking, AC^1

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.368

1 Introduction

Intuitionistic logic (see e.g. [8, 20]) is a part of classical logic that can be proven using constructive proofs–e.g. by proofs that do not use *reductio ad absurdum*. For example, the law of the excluded middle $a \vee \neg a$ and the weak law of the excluded middle $\neg a \vee \neg\neg a$ do not have a constructive proof and are not valid in intuitionistic logic. Not surprisingly, constructivism has its costs. Whereas the tautology problem is coNP-complete for classical propositional logic [5], for intuitionistic propositional logic $\mathcal{IPC}$ it is PSPACE-complete [17, 18]. The computational hardness of intuitionistic logic is already reached with the fragment $\mathcal{IPC}_2$ of formulas having only two variables: the tautology problem is PSPACE-complete already for $\mathcal{IPC}_2$ [16]. Recall that every fragment of classical propositional logic with a fixed number of variables has an NC^1-complete tautology problem (follows from [2]).

In this paper, we consider the complexity of intuitionistic propositional logic with one or two variables. The model checking problem—i.e. the problem to determine whether a given formula is satisfied by a given Kripke model—was recently shown to be P-complete [12] for $\mathcal{IPC}$. We show, that it remains P-complete for the fragment with two variables $\mathcal{IPC}_2$. More surprisingly, for the fragment with one variable $\mathcal{IPC}_1$ we show the model checking problem to be AC^1-complete. A basic ingredient for this result lies in normal forms for models and formulas as found by Nishimura [14], that we reinvestigate under an algorithmic and complexity theoretical point of view. To our knowledge, this is the first "natural" AC^1-complete problem, whereas formerly known AC^1-complete problems (see e.g. [1]) have some explicit logarithmic bound in the problem definition. In contrast, the formula value problem for classical propositional logic is NC^1-complete [2], even with one variable (follows from [2]).

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 368–379
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

Classical propositional logic is the extension of $\mathcal{IPC}$ with the axiom $a \vee \neg a$. Those proper extensions of intuitionistic logic are called superintuitionistic logics. The superintuitionistic logic $\mathcal{KC}$ (see [7]) results from adding $\neg a \vee \neg\neg a$ to $\mathcal{IPC}$. We show that the model checking problem for every superintuitionistic logic with one variable is NC^1-complete (and easier than that for $\mathcal{IPC}_1$), whereas for the superintuitionistic logic $\mathcal{KC}$ with two variables it is already P-complete (and as hard as for $\mathcal{IPC}_2$).

We also consider the tautology problem that is known to be PSPACE-complete for $\mathcal{IPC}_2$ [16]. Svejdar [19] recently showed the upper bound $\mathsf{SPACE}(\log n \cdot \log\log n)$ for $\mathcal{IPC}_1$. We show the tautology problem to be in NC^1 for any superintuitionistic logic with one variable. For superintuitionistic logics with more than one variable such a general result for the tautology problem remains open.

This paper is organized as follows. In Section 2 we introduce the notations we use for intuitionistic logic and model checking. Section 3 is devoted to introduce the old results by Nishimura [14] and to upgrade them with a complexity analysis. The following Section 4 presents our lower and upper bound for model checking for $\mathcal{IPC}_1$. Section 5 deals with the complexity of the model checking problem and the tautology problem for superintuitionistic logics with one variable, and Section 6 considers the case with two variables. The implied completeness for the model checking for intuitionistic logic and conclusions are drawn in Section 7. Proofs and technical details can be found in [13].

2 Preliminaries

Complexity (see e. g. [21]). The notion of reducibility we use is the logspace many-one reducibility $\leq_{\mathrm{m}}^{\log}$, except for NC^1 hardness, where we use first-order reducibility. NC^1 and AC^1 are the classes of sets that are decided by families of logspace-uniform circuits of polynomial size and logarithmic depth. The circuits consist of and-, or-, and negation-gates. The negation-gates have fan-in 1. For NC^1, the and- and or-gates have fan-in 2 (bounded fan-in), whereas for AC^1 there is no bound on the fan-in of the gates (unbounded fan-in). $\mathsf{ALOGTIME}$ denotes the class of sets decided by alternating Turing machines in logarithmic time, and we will use that $\mathsf{NC}^1 = \mathsf{ALOGTIME}$ (see [15]). L denotes the class of sets decidable in logarithmic space. We use $\mathsf{ALOGSPACE}[f(n)]$ to denote the class of sets decided by an alternating log-space Turing machine that makes $O(f(n))$ alternations, where n is the length of the input. We will use that $\mathsf{AC}^1 = \mathsf{ALOGSPACE}[\log n]$ (see [6]). $\mathsf{LOGDCFL}$ is the class of sets that are $\leq_{\mathrm{m}}^{\log}$-reducible to deterministic context-free languages. It is also characterized as the class of sets decidable by deterministic Turing machines in polynomial-time and logarithmic space with additional use of a stack. The inclusion structure of the classes we use is as follows.

$$\mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{LOGDCFL} \subseteq \mathsf{AC}^1 \subseteq \mathsf{P} \subseteq \mathsf{PSPACE}$$

Intuitionistic Propositional Logic (see e.g.[20]). Let VAR denote a countable set of *variables*. The language $\mathcal{IPC}$ of intuitionistic propositional logic is the same as that of propositional logic $\mathcal{PC}$, i.e. it is the set of all formulas of the form

$$\varphi ::= p \mid \bot \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \to \varphi)\,,$$

where $p \in \mathrm{VAR}$. The languages $\mathcal{IPC}_i$ are the subsets/fragments of $\mathcal{IPC}$ for which VAR consists of i variables. We will consider $\mathcal{IPC}_0$ (i.e. formulas without variables), $\mathcal{IPC}_1$ and $\mathcal{IPC}_2$ (i.e. formulas with one resp. two variables).

As usual, we use the abbreviations $\neg\varphi := \varphi \to \bot$ and $\top := \neg\bot$. Because of the semantics of intuitionistic logic, one cannot express $\wedge$ or $\vee$ using $\to$ and $\bot$.

A *Kripke model* for intuitionistic logic is a triple $\mathcal{M} = (U, R, \xi)$, where U is a nonempty and finite set of *states*, R is a preorder on U (i.e. a reflexive and transitive binary relation), and $\xi : \mathrm{VAR} \to \mathfrak{P}(U)$ is a function — the *valuation function*. Informally speaking, for any variable it assigns the set of states in which this variable is satisfied. The valuation function ξ is monotone in the sense that for every $p \in \mathrm{VAR}$, $a, b \in U$: if $a \in \xi(p)$ and aRb, then $b \in \xi(p)$. (U, R) can also be seen as a directed graph. We will call such models *intuitionistic*.

Given an intuitionistic model $\mathcal{M} = (U, \leqslant, \xi)$ and a state $s \in U$, the *satisfaction relation for intuitionistic logics* $\models$ is defined as follows.

$$\begin{array}{lll}
\mathcal{M}, s \not\models \bot & & \\
\mathcal{M}, s \models p & \text{iff} & s \in \xi(p),\ p \in \mathrm{VAR}, \\
\mathcal{M}, s \models \varphi \wedge \psi & \text{iff} & \mathcal{M}, s \models \varphi \text{ and } \mathcal{M}, s \models \psi, \\
\mathcal{M}, s \models \varphi \vee \psi & \text{iff} & \mathcal{M}, s \models \varphi \text{ or } \mathcal{M}, s \models \psi, \\
\mathcal{M}, s \models \varphi \to \psi & \text{iff} & \forall n \geqslant s : \text{ if } \mathcal{M}, n \models \varphi \text{ then } \mathcal{M}, n \models \psi
\end{array}$$

A formula φ is *satisfied* by an intuitionistic model $\mathcal{M}$ in state s iff $\mathcal{M}, s \models \varphi$. A *tautology* resp. a *valid formula* is a formula that is satisfied by every model.

The Model Checking Problem. This paper examines the complexity of model checking problems for intuitionistic logics.

Problem:	model checking problem for $\mathcal{IPC}_i$
Input:	$\langle \varphi, \mathcal{M}, s \rangle$, where φ is an $\mathcal{IPC}_i$ formula, $\mathcal{M}$ is an intuitionistic Kripke model, and s is a state of $\mathcal{M}$
Question:	$\mathcal{M}, s \models \varphi$?

We assume that formulas and Kripke models are encoded in a straightforward way. This means, a formula is given as a text, and the graph (U, R) of a Kripke model is given by its adjacency matrix that takes $|U|^2$ bits. Therefore, only finite Kripke models can be considered.

3 Properties of $\mathcal{IPC}_1$ and its complexity

The set $\mathcal{IPC}_1$ of formulas with one variable is partitioned into infinitely many equivalence classes [14]. This was shown using the formulas that are inductively defined as follows (see e.g.[8]). We use a for the only variable.

$$\varphi_1 = \neg a, \qquad \psi_1 = a, \qquad \varphi_{n+1} = \varphi_n \to \psi_n, \qquad \psi_{n+1} = \varphi_n \vee \psi_n$$

The formulas $\bot, \top, \varphi_1, \psi_1, \varphi_2, \psi_2, \ldots$ are called *Rieger-Nishimura formulas.*

▶ **Theorem 1.** *([14], cf.[8, Chap.6,Thm.7]) Every formula in $\mathcal{IPC}_1$ is equivalent*[1] *to exactly one of the Rieger-Nishimura formulas.*

The function *RNindex* maps every formula to the index of its equivalent Rieger-Nishimura formula.

$$RNindex(\alpha) = \begin{cases} (i, phi), & \text{if} \quad \alpha \equiv \varphi_i \\ (i, psi), & \text{if} \quad \alpha \equiv \psi_i \\ (0, \bot), & \text{if} \quad \alpha \equiv \bot \\ (0, \top), & \text{if} \quad \alpha \equiv \top \end{cases}$$

[1] α is equivalent to β if every state in every model satisfies both or none formula.

We analyze the complexity of this function in Lemma 3. For this, we need a lower bound for the length of formulas in every equivalence class (see Lemma 2). With other words, this is an upper bound of the length of the Rieger-Nishimura index of any formula.

Let $[\varphi]$ denote the equivalence class that contains φ, for φ being an $\mathcal{IPC}_1$ formula. The equivalence classes of the $\mathcal{IPC}_1$ formulas form a free Heyting algebra over one generator (for algebraic details see [10]). This algebra is also called the *Rieger-Nishimura lattice* (see Fig. 1). It is shown in [14] that the lattice operations can be calculated using a big case distinction (see [13, Appendix A]). We use these algebraic properties of $\mathcal{IPC}_1$ to give a lower bound on the length of formulas[2] in the equivalence classes of $\mathcal{IPC}_1$, and to give an upper bound on the complexity of the problem to decide the Rieger-Nishimura index of a formula. Let $\mathit{rank}(\alpha)$ be the first element—the integer—of the $\mathit{RNindex}(\alpha)$ pair.

▶ **Lemma 2.** *For every $\mathcal{IPC}_1$ formula φ it holds that $\mathit{rank}(\varphi) \leq c_1 \cdot \log(|\varphi|)$, for a constant c_1 independent of φ.*

In order to analyze the complexity of the Rieger-Nishimura index computation, we define the following decision problem.

Problem: EqRNformula
Input: $\langle \alpha, (i, x) \rangle$, where α is an $\mathcal{IPC}_1$ formula and (i, x) is an index
Question: Is (i, x) the Rieger-Nishimura index of α?

▶ **Lemma 3.** EqRNformula *is in* LOGDCFL.

Similar as any formula can be represented by its index, Kripke models can, too. We give a construction of models—the *canonical models*—(according to [8, Chap.6, Defi.5]) that are also used to distinguish the formula equivalence classes (Theorem 4). Our definition is a little different from that in [8, Chap.6, Defi.5]. We show in Lemma 5 that every model over one variable is homomorphic to a canonical model, and that this homomorphic Kripke model can be determined in $\mathsf{ALOGSPACE}[n]$. (For model checking it suffices to determine at most $\log n$ as index. For details see Section 4.4.) For $n = 1, 2, \ldots$, we define the canonical models $\mathcal{H}_n = (W_n, \trianglelefteq, \xi_n)$ as follows (according to [8, Chap.6, Defi.5]).

1. $W_n = \{1, 2, \ldots, n-2\} \cup \{n\}$,
2. $a \trianglelefteq b$ iff $a = b$ or $a \geq b + 2$, and
3. $\xi_n(a) = \begin{cases} \emptyset, & \text{if } n = 2 \\ \{1\}, & \text{otherwise.} \end{cases}$

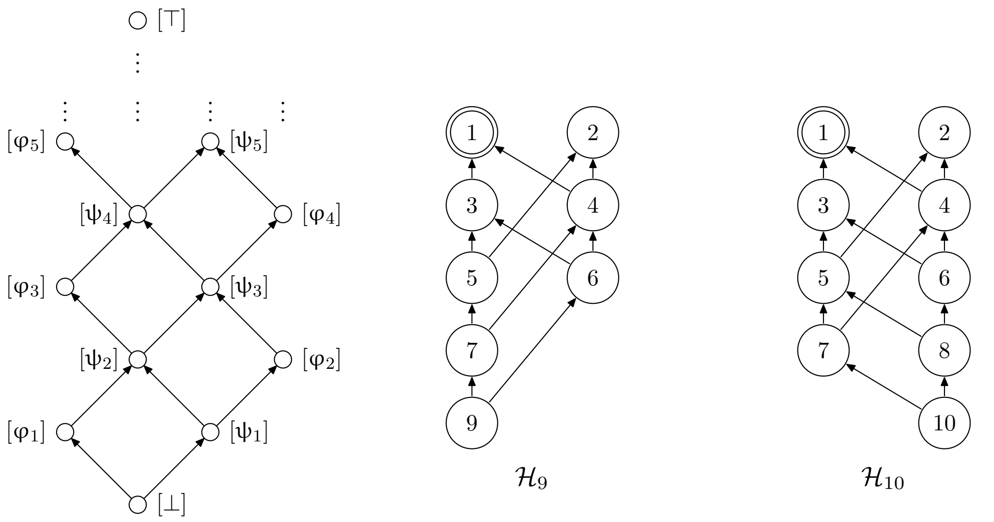
See Figure 1 for some examples.

The formulas in $\mathcal{IPC}_1$ can be distinguished using the canonical models as follows.

▶ **Theorem 4.** *([14],cf.[8, Chap.6, Thm.8]) For every n and every k holds:*
1. *$\mathcal{H}_n, n \models \psi_k$ iff $n \leq k$ (i.e. $k \in \{n, n+1, \ldots\}$), and*
2. *$\mathcal{H}_n, n \models \varphi_k$ iff $n < k$ or $n = k + 1$ (i.e. $k \in \{n-1\} \cup \{n+1, n+2, \ldots\}$).*

From [8, Chap.6, Lemma 11] it is known that every intuitionistic Kripke model $\mathcal{M}$ (for formulas with one variable a) is homomorphic to some $\mathcal{H}_i$. We additionally analyse the complexity of the decision problem whether $\mathcal{M}$ is homomorphic to $\mathcal{H}_i$ (see Lemma 5). For

[2] $|\alpha|$ denotes the length of the formula α, and it is the number of appearances of variables, connectives, and constants in α.



The Rieger-Nishimura lattice

Figure 1 The Rieger-Nishimura lattice (left), and the canonical models $\mathcal{H}_9$ and $\mathcal{H}_{10}$ (reflexive and transitive edges are not depicted, $\xi_n(a) = \{1\}$ is indicated by the double circle for state 1).

this, we define a function $\hbar$ that for given model $\mathcal{M}$ and state w has as function value $\hbar(\mathcal{M}, w)$ the index i of the homomorphic model $\mathcal{H}_i$. Let $\mathcal{M} = (W, \leqslant, \zeta)$ be a model and w a state. Let $W_{w\Uparrow} = \{v \in W \mid w \leqslant v\}$ and $W_{w\uparrow} = W_{w\Uparrow} \setminus \{w\}$. We define $\hbar$ as follows.

$$\hbar(\mathcal{M}, w) = \begin{cases} 1, & \text{if} \quad w \in \zeta(a) \\ 2, & \text{if} \quad w \notin \zeta(a) \text{ and } \forall v \in W_{w\uparrow} \; v \notin \zeta(a) \\ 3, & \text{if} \quad w \notin \zeta(a) \text{ and } \forall v \in W_{w\uparrow} \; \hbar(\mathcal{M}, v) \neq 2 \text{ and } \exists u \in W_{w\uparrow} \; \hbar(\mathcal{M}, u) = 1 \\ n+2, & \text{if} \quad \forall v \in W_{w\uparrow} \; \hbar(\mathcal{M}, v) \neq n+1 \text{ and} \\ & \qquad\qquad \exists u_1, u_2 \in W_{w\uparrow} \; \hbar(\mathcal{M}, u_1) = n \text{ and } \hbar(\mathcal{M}, u_2) = n-1 \end{cases}$$

We call $\hbar(\mathcal{M}, w)$ the *model index* of w. The function $\hbar$ is well defined because for every state w it holds that $\{\hbar(\mathcal{M}, v) \mid v \in W_{w\Uparrow}\} = \{1, 2, \ldots, \hbar(\mathcal{M}, w) - 2\} \cup \{\hbar(\mathcal{M}, w)\}$.

Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be models, w_1 resp. w_2 a state from $\mathcal{M}_1$ resp. $\mathcal{M}_2$. We say that $(\mathcal{M}_1, w_1)$ is *homomorphic* to $(\mathcal{M}_2, w_2)$ if for every $\alpha \in \mathcal{IPC}_1$ it holds that $\mathcal{M}_1, w_1 \models \alpha$ iff $\mathcal{M}_2, w_2 \models \alpha$.

▶ **Lemma 5.** *Let $\mathcal{M} = (W, \leq, \zeta)$ be a Kripke model.*

1. *$(\mathcal{M}, w)$ is homomorphic to $(\mathcal{H}_{\hbar(\mathcal{M},w)}, \hbar(\mathcal{M}, w))$.*
2. *For a given model $\mathcal{M}$, a state w and an integer n, to decide whether $\hbar(\mathcal{M}, w) = n$ is in* ALOGSPACE$[n]$.

4 The complexity of model checking for $\mathcal{IPC}_1$

We first define an AC^1-hard graph problem, that is similar to the alternating graph accessibility problem, but has some additional simplicity properties. Then we give a construction that transforms such a graph into an intuitionistic Kripke model. This transformation is the basis for the reduction from the alternating graph accessibility problem to the model checking problem for $\mathcal{IPC}_1$.

4.1 Alternating graph problems

The alternating graph accessibility problem is shown to be P-complete in [4]. We use the following restricted version of this problem that is very similar to Boolean circuits with and- and or-gates (and input-gates). An *alternating slice graph* [12] $G = (V, E)$ is a directed bipartite acyclic graph with a bipartitioning $V = V_\exists \cup V_\forall$, and a further partitioning $V = V_0 \cup V_1 \cup V_2 \cup \cdots \cup V_{m-1}$ (m *slices*, $V_i \cap V_j = \emptyset$ if $i \neq j$) where $V_\exists = \bigcup_{i<m, i \text{ odd}} V_i$ and $V_\forall = \bigcup_{i<m, i \text{ even}} V_i$, such that $E \subseteq \bigcup_{i=1,2,\ldots,m-1} V_i \times V_{i-1}$ — i.e. all edges go from slice V_i to slice V_{i-1} (for $i = 1, 2, \ldots, m-1$). All nodes excepted those in the last slice V_0 have a positive outdegree. Nodes in $V_\exists$ are called *existential* nodes, and nodes in $V_\forall$ are called *universal* nodes. Alternating paths from node x to node y are defined as follows by the property $apath_G(x, y)$.

1) $apath_G(x, x)$ holds for all $x \in V$

2a) for $x \in V_\exists$: $apath_G(x, y)$ iff $\exists z \in V_\forall : (x, z) \in E$ and $apath_G(z, y)$

2b) for $x \in V_\forall$: $apath_G(x, y)$ iff $\forall z \in V_\exists :$ if $(x, z) \in E$ then $apath_G(z, y)$

The problem AsAgap is similar to the alternating graph accessibility problem, but for the restricted class of alternating slice graphs.

Problem: AsAgap
Input: $\langle G, s, t\rangle$, where $G = (V_\exists \cup V_\forall, E)$ is an alternating slice graph with slices $V_0, V_1, \ldots, V_{m-1}$, and $s \in V_{m-1} \cap V_\exists$, $t \in V_0 \cap V_\forall$
Question: does $apath_G(s, t)$ hold?

Similarly as the alternating graph accessibility problem, AsAgap is P-complete [12, Lemma 2]. The following technical Lemma is not hard to prove.

▶ **Lemma 6.** *For every set A in (logspace-uniform)* AC^1 *exists a function f that maps instances x of A to instances $f(x) = \langle G_x, s_x, t_x\rangle$ of* AsAgap *and satisfies the following properties.*

1. *f is computable in logspace.*
2. *G_x is an alternating slice graph of logarithmic depth; i.e. if G_x has n nodes, then it has $m \leq \log n$ slices.*
3. *For all instances x of A holds: $x \in A$ iff $f(x) \in$* AsAgap.

Essentially, the function f constructs the AC^1 circuit $C_{|x|}$ with input x, and transforms it to an alternating slice graph G_x. The goal node t_x represents exactly the bits of x that are 1. The start node s_x corresponds to the output gate of $C_{|x|}$, and $apath_{G_x}(s_x, t_x)$ expresses that $C_{|x|}$ accepts input x.

4.2 Transforming alternating slice graphs to intuitionistic Kripke models

Our hardness results rely on a transformation of instances $\langle G, s, t\rangle$ of AsAgap to Kripke models $\mathcal{M}_G = (U, R, \xi)$. We describe it informally here. An example can be seen in Figure 2.

Let $\langle G, s, t\rangle$ be an instance of AsAgap for the slice graph $G = (V_\exists \cup V_\forall, E_G)$ with the m slices $V_\exists = V_{m-1} \cup V_{m-3} \cup \cdots \cup V_1$ and $V_\forall = V_{m-2} \cup V_{m-4} \cup \cdots \cup V_0$. The Kripke model $\mathcal{M}_G$ is constructed as follows. The nodes of the Kripke model consist of the nodes of $\mathcal{H}_{4m}$, and of two copies u^{in} and u^{out} of each node u of G. The slice $\mathcal{W}_i^{out}$ of $\mathcal{M}_G$ consists of the *out*-copies of nodes in V_i and the nodes $4i + 1$ and $4i + 2$ of $\mathcal{H}_{4m}$, and the slice $\mathcal{W}_i^{in}$ of $\mathcal{M}_G$ consists of the *in*-copies of nodes in V_i and the nodes $4i + 3$ and $4i + 4$ of $\mathcal{H}_{4m}$.

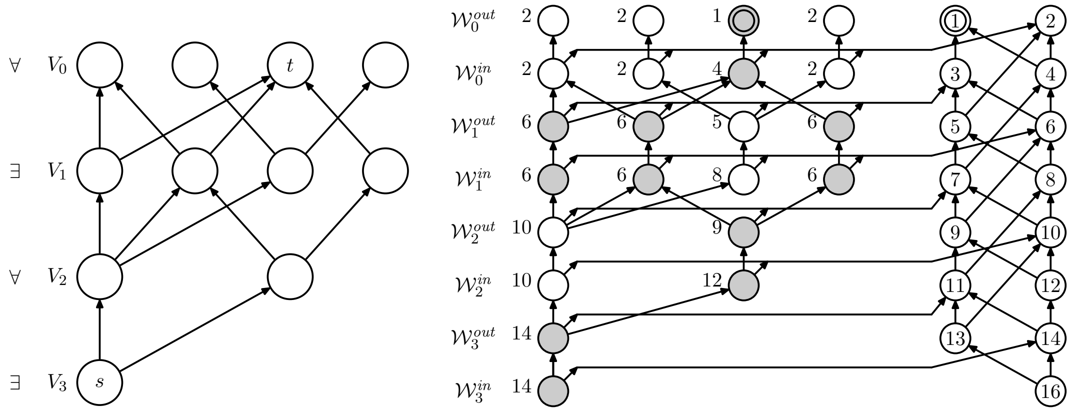


Figure 2 An alternating slice graph G (left) and the resulting Kripke model $\mathcal{M}_G$ (right); both the states in $\xi(a)$ are drawn doubly; pseudotransitive and reflexive edges in $\mathcal{M}_G$ are not depicted. The value at state x denotes its model index $\hslash(\mathcal{M}_G, x)$. For states in $\mathcal{H}_{16}$, their names and their model indices coincide. States v^{in} and v^{out} for which $apath_G(v,t)$ holds in G are colored grey.

All the edges of $\mathcal{H}_{4m}$ are kept. The edges (u,v) of G are changed to (u^{out}, v^{in}) in $\mathcal{M}_G$, and for every u of G an edge (u^{in}, u^{out}) is added. We add edges from the copies of the nodes in G to the nodes from $\mathcal{H}_{4m}$ as follows. Every node v^{out} for $v \in V_i$ $(i > 0)$ has an edge to node $4i - 1$ from $\mathcal{H}_{4m}$, and every node v^{in} for $v \in V_i$ has an edge to node $4i + 2$ from $\mathcal{H}_{4m}$.

An intuitionistic Kripke model must be transitive and reflexive. The part of the model that comes from $\mathcal{H}_{4m}$ is transitive and reflexive, the part of the model that comes from the alternating slice graph G is not. The transformation of the alternating slice graph to an intuitionistic Kripke model must be computable in logarithmic space, because it will be used as part of a logspace reduction function. Within this space bound we cannot compute the transitive closure of a graph. Therefore, we make the graph transitive with brute force. We add all edges from nodes v^{in} and v^{out} $(v \in V)$ that jump over at least one slice—we call these edges *pseudotransitive*. Finally, we need to add all missing reflexive edges.

The valuation function for our model $\mathcal{M}_G$ is $\xi(a) = \{t^{out}, 1\}$, where t^{out} is the copy of the goal node t in $\mathcal{W}_0^{out}$, and $\{1\} = \xi_{4m}(a)$ is the node from $\mathcal{H}_{4m}$. This concludes the informal description of the Kripke model $\mathcal{M}_G$. An example of an AsAgap instance $\langle G, s, t\rangle$ and the Kripke model $\mathcal{M}_G$ constructed from it can be seen in Figure 2.

The canonical model is attached to the slice graph in order to obtain control over the model indices of the other states (w.r.t. the model $\mathcal{M}_G$). This is described by Propositions 7 and 8.

▶ Proposition 7. For every $i = 0, 1, 2, \ldots, m-1$ and every $v \in V_i$ holds

$$\hslash(\mathcal{M}_G, v^{out}) \in \{4i+1, 4i+2\} \quad \text{and} \quad \hslash(\mathcal{M}_G, v^{in}) \in \{4i+2, 4i+4\} \enspace .$$

▶ Proposition 8. For every $i = 0, 1, 2, \ldots, m-1$ and every $v \in V_i$ holds:

1. if i is even (∀ slice):
   $apath_G(v,t)$ iff $\hslash(\mathcal{M}_G, v^{out}) = 4i+1$, and $apath_G(v,t)$ iff $\hslash(\mathcal{M}_G, v^{in}) = 4i+4$,
2. if i is odd (∃ slice):
   $apath_G(v,t)$ iff $\hslash(\mathcal{M}_G, v^{out}) = 4i+2$, and $apath_G(v,t)$ iff $\hslash(\mathcal{M}_G, v^{in}) = 4i+2$.

Let T denote the function that maps instances $x = \langle G, s, t\rangle$ of AsAgap to Kripke models $T(x) = \mathcal{M}_G$ as described above. The following properties of T are easy to verify.

▶ **Lemma 9.** *1. T is logspace computable.*

2. If $x = \langle G, s, t\rangle$ for an alternating slice graph G with n nodes and $m < n$ slices, then $T(x)$ is a Kripke model with $\leq 4n$ states and depth $2m$.

We will use T as part of the reduction functions for our hardness results.

4.3 Hardness results

Our first result states that the calculation of the model index of an intuitionistic Kripke model is P-complete. It is already P-complete to decide the last bit of this model index.

▶ **Theorem 10.** *The following problems are* P*-complete.*

1. *Given a Kripke model $\mathcal{M}$ and a state w, decide whether $\hbar(\mathcal{M}, w)$ is even.*
2. *Given a Kripke model $\mathcal{M}$, a state w, and an integer i, decide whether $\hbar(\mathcal{M}, w) = i$.*

Proof. In order to show the P-hardness of the problems, we give a reduction from the P-hard problem AsAgap [12]. From an instance $\langle G, s, t\rangle$ of AsAgap where G is an alternating slice graph with m slices, construct $\mathcal{M} = T(\langle G, s, t\rangle)$. Then $\hbar(\mathcal{M}, s^{out}) \in \{4m + 1, 4m + 2\}$ (Proposition 7), and $apath_G(s, t)$ iff $\hbar(\mathcal{M}, s^{out}) = 4m+2$ (Proposition 8). Therefore, $\langle G, s, t\rangle \in$ AsAgap if and only if $\hbar(\mathcal{M}, s^{out})$ is even resp. $\hbar(\mathcal{M}, s^{out}) = 4m + 2$.

For every model $\mathcal{M} = (U, \leqslant, \xi)$ holds $\hbar(\mathcal{M}, w) \leq |U| + 1$. From Lemma 5 and using that $\mathsf{P} = \mathsf{ALOGSPACE}[poly]$ it then follows that both problems are in P. ◀

In the construction of the above proof, the decision whether $\hbar(\mathcal{M}, s^{out}) = 4m + 2$ is the same as to decide whether $\mathcal{M}, s^{out} \models \psi_{4m+2}$, for the Rieger-Nishimura formula ψ_{4m+2}. Unfortunately, the length of ψ_{4m+2} is exponential in m, and therefore the mapping from $\langle G, s, t\rangle$ (with m slices) to the model checking instance $\langle \psi_{4m+2}, T(\langle G, s, t\rangle), s^{out}\rangle$ cannot in general be performed in logarithmic space. But if the depth m of the slice graph is logarithmic, the respective formula ψ_{4m+2} has polynomial size only. Using Lemma 6, every AC^1 problem is logspace reducible to instances of AsAgap having logarithmically bounded depth, and by the above mapping these special instances are logspace reducible to model checking for $\mathcal{IPC}_1$.

▶ **Theorem 11.** *The model checking problem for $\mathcal{IPC}_1$ is AC^1-hard.*

4.4 Model checking for $\mathcal{IPC}_1$ is in AC^1

Algorithm 1 decides the model checking problem for $\mathcal{IPC}_1$. In the following, we estimate its complexity. The algorithm gets input $\langle \varphi, \mathcal{M}, s\rangle$ and works in two steps. In the first step we calculate the Rieger-Nishimura index (r, x) of φ. Since the index is very small (Lemma 2) and using Lemma 3, this can be done in LOGDCFL. In the second step we identify the homomorphic canonical model for $(\mathcal{M}, s)$. In fact it is not always necessary to identify the homomorphic canonical model exactly. According to Theorem 4, the input can be rejected if $\hbar(\mathcal{M}, s) > r + 1$, and the latter can be checked by finding some state $u \geqslant s$ in $\mathcal{M}$ with $\hbar(\mathcal{M}, u) > r + 1$. This estimation can be done in alternating logarithmic space with $r + 1$ alternations. If $\hbar(\mathcal{M}, u) \leq r + 1$, according to Theorem 5 the model index can be computed exactly in $\mathsf{ALOGSPACE}[r + 1]$, and knowing the Rieger-Nishimura index of φ and the model index of $(\mathcal{M}, s)$, it can be decided whether $\mathcal{M}, s \models \varphi$ using Theorem 4. Since k is at most about $\log |\varphi|$ (Lemma 2), the computation of the model index can be done in alternating logspace with $\log |\langle \varphi, \mathcal{M}, s\rangle|$ alternations (Lemma 5). Since the Rieger-Nishimura index of a formula can be decided in LOGDCFL, and $\mathsf{LOGDCFL} \subseteq \mathsf{AC}^1 = \mathsf{ALOGSPACE}[\log n]$, we obtain the desired upper bound.

Algorithm 1 model checking algorithm for $\mathcal{IPC}_1$

```
Require: a formula $\varphi$, a model $\mathcal{M}$ and a state $s$
search for $(r, x)$ with $0 \leq r \leq c_1 \cdot \log(|\varphi|)$ such that $\langle \varphi, (r, x) \rangle \in$ EQRNFORMULA
if $(r, x) = (0, \bot)$ then reject
else if $(r, x) = (0, \top)$ then accept
else if $x = psi$ then
  if $\hbar(\mathcal{M}, s) \neq i$ for all $i \in \{1, 2, \dots, r\}$ then reject
  else accept
else if $x = phi$ then
  if $\hbar(\mathcal{M}, s) \neq i$ for all $i \in \{1, 2, \dots, r-1\} \cup \{r+1\}$ then reject
  else accept
end if
```

▶ **Theorem 12.** *The model checking problem for* $\mathcal{IPC}_1$ *is in* AC^1.

5 Some notes on superintuitionistic logics with one variable

Superintuitionistic propositional logics are logics that have more valid formulas than $\mathcal{IPC}$. In this sense, classical propositional logic is a superintuitionistic logic, since it can be obtained as the closure under substitution and modus ponens of the tautologies from $\mathcal{IPC}$ plus $a \vee \neg a$ as additional axiom. A well-studied superintuitionistic logic is $\mathcal{KC}$ [7] that results from adding the weak law of the excluded middle $\neg a \vee \neg\neg a$ to $\mathcal{IPC}$. Semantically, the Kripke models for $\mathcal{KC}$ are restricted to those intuitionistic models $\mathcal{M} = (W, \leqslant, \xi)$ where $\leqslant$ is a *directed* preorder. Whereas $\mathcal{IPC}_1$ has infinitely many equivalence classes of formulas, $\mathcal{KC}_1$ has only 7 equivalence classes—represented by the Rieger-Nishimura formulas $\bot, \top, \varphi_1, \psi_1, \varphi_2, \psi_2, \psi_3$—that can be distinguished using the first 3 canonical models [14, 11]. The function $\hbar$ can be implemented as an alternating Turing machine that runs in logarithmic time, if the function value is fixed to a finite range—that in this case is $\{1, 2, 3\}$–independent on the input. For $\mathcal{KC}_1$, the Rieger-Nishimura index of the formulas also has a finite range (as mentioned above). Therefore, it can be calculated by an alternating Turing machine that runs in logarithmic time similar to the machine presented by Buss [3] that calculates the value of a Boolean formula. Instead of the Boolean values 0 and 1, for $\mathcal{KC}_1$ we have 7 different Rieger-Nishimura indices. The rules how the index of a formula can be calculated from the indices of its subformulas and the connective, follow directly from the Rieger-Nishimura lattice operations—see e.g. [13]. If the indices are bound to a finite range, this big table yields an even bigger but finite table without variable formula indices. For example, the equivalence $\varphi_n \vee \varphi_{n+1} \equiv \psi_{n+2}$ for all $n \geq 1$ induces the three equivalences $\varphi_1 \vee \varphi_2 \equiv \psi_3$, $\varphi_2 \vee \top \equiv \top$, and $\top \vee \top \equiv \top$ for $\mathcal{KC}_1$. This yields $\mathsf{ALOGTIME}$ as an upper bound for the tautology problem for $\mathcal{KC}_1$.

There are infinitely many superintuitionistic logics (with one variable) that can be obtained by adding an arbitrary formula as axiom to $\mathcal{IPC}_1$, that is not valid for $\mathcal{IPC}_1$. For example, if we add a formula equivalent to φ_k, then the superintuitionistic logic obtained has finitely many equivalence classes represented by $\bot, \top, \varphi_1, \psi_1, \varphi_2, \psi_2, \dots, \varphi_{k-1}, \psi_{k-1}$. With similar arguments as for $\mathcal{KC}_1$ we can conclude that the model checking problems for these logics all are in NC^1. Moreover, the formula value problem for Boolean formulas without variables is NC^1-hard [2]. Intuitionistic formulas without variables have the same values, if they are interpreted as classical Boolean formulas. This means, the semantics of → is the same for Boolean formulas and for intuitionistic formulas without variables. Therefore, the model checking problem for any superintuitionistic logic is NC^1-hard, too.

▶ **Theorem 13.** *The model checking problem for every superintuitionistic logic with one variable is* NC^1*-complete.*

The tautology problem for superintuitionistic logic has the same complexity, since in order to decide whether a formula with one variable is a tautology it suffices to know its Rieger-Nishimura index.

▶ **Theorem 14.** *The tautology problem for every superintuitionistic logic with one variable is* NC^1*-complete.*

The best upper space bound for the tautology problem for $\mathcal{IPC}_1$ is a little higher, namely $\mathsf{SPACE}(\log n \cdot \log\log n)$ [19].

6 The complexity of model checking for $\mathcal{IPC}_2$

▶ **Theorem 15.** *The model checking problems for* $\mathcal{KC}_2$ *and for* $\mathcal{IPC}_2$ *are* P*-hard.*

This can be proven with a reduction from the $\mathcal{IPC}$ model checking problem that is P-complete [12]. We give formulas with two variables—the *replacement formulas*—which simulate the variables in an arbitrary $\mathcal{IPC}$ formula. For an arbitrary model we need to simulate the assignment function. For this we use a special model where every replacement formula has a unique maximal refuting state. We combine the given model and the special model in a way that if a variable is not assigned to a state, this state is connected to the state from the special model that refutes the replacement formula which substitutes this variable. This construction is similar to the polynomial time reduction from the tautology problem for $\mathcal{IPC}$ to the tautology problem for $\mathcal{IPC}_2$ in [16]. The main difference is that our logspace reduction yields pseudotransitive models, whereas in [16] a polynomial time reduction is used that allows to compute transitive models.

7 Conclusion

We consider computational problems that appear with intuitionistic propositional logic with at most two variables. Our main theorem completely characterizes the complexity of model checking for intuitionistic logic.

▶ **Theorem 16.**
1. *The model checking problem for* $\mathcal{IPC}_0$ *is* NC^1*-complete.*
2. *The model checking problem for* $\mathcal{IPC}_1$ *is* AC^1*-complete.*
3. *The model checking problem for* $\mathcal{IPC}_2$ *is* P*-complete.*

Part (1) follows from the fact that an intuitionistic formula that contains constants $\bot$ and $\top$ but no variables can be evaluated like a Boolean formula, whose evaluation problem is NC^1-complete [2] independently of the number of variables. Part (2) follows from Theorems 11 and 12. It shows a difference between $\mathcal{IPC}_1$ and its modal companion $\mathcal{S}4$ with one variable, for which the model checking problem is P-complete [12]. Part (3) comes from Theorem 15 (hardness) and [12] (containment in P). Similarly as Rybakov [16] for the tautology problem, this shows that the model checking problem for $\mathcal{IPC}$ reaches its full complexity already with the use of two variables.

Next we summarize the results from Theorems 13 and 15 for superintuitionistic logics.

1. *The model checking problem for every superintuitionistic logic with one variable is* NC^1*-complete.*
2. *The model checking problem for* $\mathcal{KC}_2$ *is* P*-complete.*

There are superintuitionistic logics with two variables between Boolean logic and $\mathcal{KC}_2$ (see below). The exact complexity of the model checking problem for these logics is open. It is interesting to notice that the complexity results for $\mathcal{IPC}_2$ and for $\mathcal{KC}_2$ are the same. But if only one variable is allowed, the complexity of $\mathcal{IPC}_1$ is higher than that of $\mathcal{KC}_1$.

Intuitionistic logic with one variable turns out to be the most challenging case. There are infinitely many equivalence classes of formulas, and according to Lemma 2 the sequence of smallest representatives of these equivalence classes has an exponential growth with respect to the length of the formulas. Such a fast growing sequence seems to appear rarely in "natural" problems, and it is a key ingredient for the AC^1-completeness of the model checking problem. Intuitionistic logic with one variable is strongly related to free Heyting algebras with one generator. Since Heyting algebras are generalizations of Boolean algebras, it would be interesting to investigate whether the difference between NC^1 and AC^1 is related to that between Boolean algebras and Heyting algebras.

If we consider other problems related to Kripke models for $\mathcal{IPC}_1$ that are not "out braked" by a very fast growing part of the input, the complexity jumps up to P-completeness, as shown in Theorem 10. Model checking for $\mathcal{IPC}_1$ also gets P-hard if the instances $\langle \varphi, \mathcal{M}, s\rangle$ allow the formula φ to be represented as a graph. Let us call this the *g-model checking problem* for short. This is a consequence of the P-hardness of the monotone circuit evaluation problem [9], holds even for formulas without variables, and therefore it also holds for all superintuitionistic logics. If formulas are represented as graphs, the sequence of smallest representatives of the equivalence classes of $\mathcal{IPC}_1$ does not have exponential growth anymore. Moreover, the calculation of the Rieger-Nishimura index gets P-hard.

▶ **Theorem 17.** *The following problems are* P*-complete:*

1. *the g-model checking problem for* $\mathcal{IPC}_1$*,*
2. *the g-model checking problem for every superintuitionistic logic with one variable,*
3. *the tautology problem for* $\mathcal{IPC}_1$*, where the formula is represented as a graph, and*
4. *the tautology problem for every superintuitionistic logic with one variable, where the formula is represented as a graph.*

Parts (1) and (2) contrast the different upper bounds NC^1 and AC^1 (Theorem 13 resp. Theorem 16) for the standard encodings of formulas. Parts (3) and (4) contrast the complexity of the tautology problems for the logics under consideration, that have the following upper bounds.

1. *The tautology problem for every superintuitionistic logic with one variable is* NC^1*-complete.*
2. *The tautology problem for* $\mathcal{IPC}_1$ *is in* $\mathsf{LOGDCFL} \cap \mathsf{SPACE}(\log n \cdot \log\log n)$*.*
3. *The tautology problem for* $\mathcal{KC}_2$ *and for* $\mathcal{IPC}_2$ *is* PSPACE*-complete.*

Part (1) is Theorem 14, part (2) is from Svejdar [19] and Lemma 3, and part (3) follows directly from [16]. The exact complexity of the tautology problem for $\mathcal{IPC}_1$ is open. It is interesting to notice that superintuitionistic logics with one variable all have lower complexity than $\mathcal{IPC}_1$, whereas for superintuitionistic logics with two variables already $\mathcal{KC}_2$ reaches the same complexity as $\mathcal{IPC}_2$. A superintuitionistic logic between Boolean logic and $\mathcal{KC}$ is $\mathcal{LC}$ [7]. Non-trivial hardness results for $\mathcal{LC}_2$ are not known. It would be interesting to investigate exactly the complexity jump from $\mathcal{LC}_2$ to $\mathcal{KC}_2$.

Acknowledgements The authors thank Vitek Svejdar, Heribert Vollmer, Johannes Süpke, Robert Zeranski, and Thomas Schneider for helpful discussions.

References

1 M. Beaudry and P. McKenzie. Circuits, matrices, and nonassociative computation. *J. Comput. Syst. Sci.*, 50(3):441–455, 1995.

2 S. R. Buss. The Boolean formula value problem is in ALOGTIME. In *Proc. 19th STOC*, pages 123–131. ACM Press, 1987.

3 S. R. Buss. Algorithms for Boolean formula evaluation and for tree contraction. In *Arithmetic, Proof Theory, and Computational Complexity*, pages 96–115. Oxford University Press, 1993.

4 A. K. Chandra, D. Kozen, and L. J. Stockmeyer. Alternation. *J. ACM*, 28:114–133, 1981.

5 S. A. Cook. The complexity of theorem proving procedures. In *Proc. 3rd STOC*, pages 151–158. ACM Press, 1971.

6 S. A. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64:2–22, 1985.

7 M. Dummett and E. Lemmon. Modal logics between S4 and S5. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 14(24):250–264, 1959.

8 D. M. Gabbay. *Semantical investigations in Heyting's intuitionistic logic.* D.Reidel, Dordrecht, Boston, London, 1981.

9 L. M. Goldschlager. The monotone and planar circuit value problems are log space complete for P. *SIGACT News*, 9(2):25–29, 1977.

10 P. T. Johnstone. *Stone spaces.* Cambridge University Press, Cambridge, 1982.

11 D. Makinson. There are infinitely many diodorean modal functions. *J. of Symbolic Logic*, 31(3):406–408, 1966.

12 M. Mundhenk and F. Weiß. The complexity of model checking for intuitionistic logics and their modal companions. In *Proc. of RP 2010*, volume 6227 of *LNCS*, pages 146–160. Springer, 2010.

13 M. Mundhenk and F. Weiß. The model checking problem for propositional intuitionistic logic with one variable is AC^1-complete. *ArXiv e-prints*, abs/1012.3828v1, 2010.

14 I. Nishimura. On formulas of one variable in intuitionistic propositional calculus. *J. of Symbolic Logic*, 25:327–331, 1960.

15 W. L. Ruzzo. On uniform circuit complexity. *Journal of Computer and Systems Sciences*, 21:365–383, 1981.

16 M. N. Rybakov. Complexity of intuitionistic and Visser's basic and formal logics in finitely many variables. In *Papers from the 6th conference on "Advances in Modal Logic"*, pages 393–411. College Publications, 2006.

17 R. Statman. Intuitionistic propositional logic is polynomial-space complete. *Theor. Comput. Sci.*, 9:67–72, 1979.

18 V. Svejdar. On the polynomial-space completeness of intuitionistic propositional logic. *Arch. Math. Log.*, 42(7):711–716, 2003.

19 V. Svejdar. The tautology problem for IPC_1 is in space $\log n \cdot \log\log n$, 2009. Personal communication.

20 D. van Dalen. *Logic and Structure.* Springer, Berlin, Heidelberg, 4th edition, 2004.

21 H. Vollmer. *Introduction to Circuit Complexity – A Uniform Approach.* Texts in Theoretical Computer Science. Springer Verlag, Berlin Heidelberg, 1999.

A Fast Algorithm for Multi-Machine Scheduling Problems with Jobs of Equal Processing Times

Alejandro López-Ortiz[1] **and Claude-Guy Quimper**[2]

1 **University of Waterloo, Waterloo, Canada**
alopez-o@uwaterloo.ca

2 **Université Laval, Québec, Canada**
claude-guy.quimper@ift.ulaval.ca

Abstract

Consider the problem of scheduling a set of tasks of length p without preemption on m identical machines with given release and deadline times. We present a new algorithm for computing the schedule with minimal completion times and makespan. The algorithm has time complexity $O(\min(1, \frac{p}{m})n^2)$ which improves substantially over the best known algorithm with complexity $O(mn^2)$.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, E.1 Data Structures

Keywords and phrases Scheduling

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.380

1 Introduction

We consider the problem of scheduling a set of equal length tasks without preemption on m identical machines with given release and deadline times. The goal is to produce a schedule, if one exists, that minimizes the sum of the completion times. We later prove that this simultaneously minimizes the makespan. This scheduling problem is known as $Pm|r_j; p_j = p; D_j| \sum C_j$ in the notation used by Pinedo [8].

The scheduling problem we study is formally defined as follows. There are n jobs labeled from 1 to n with integer release times r_i and latest starting times u_i such that $r_i < u_i$ for $i \in 1..n$. A job can start on or after time r_i but must start strickly before time u_i. Each job has an integer processing time p and needs to be allocated on one of the m identical machines. Jobs are not allowed to be preempted and only one job at a time can be executed on a machine. The deadline of job i is therefore given by $u_i + p - 1$. We therefore need to find for each job i a starting time s_i such that $r_i \leq s_i < u_i$ and that for any time t, no more than m jobs are being executed. Moreover, we would like to minimize the total completion time, i.e. the sum of the completion time of each job. Formally, we have the following system to solve.

$$\min \sum_{i=1}^{n} s_i \tag{1}$$

$$r_i \leq s_i < u_i \qquad \forall i \in 1..n \tag{2}$$

$$|\{i \mid t \leq s_i < t + p\}| \leq m \qquad \forall t \tag{3}$$

Simons [9] proposed the first polynomial time algorithm running in $O(n^3 \log\log n)$ for solving this problem. Simons and Warmuth [10] further improved this bound to $O(mn^2)$.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 380–391
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Vakhania [11] presented an algorithm that runs in $O(d_{\max}^2(m + d_{\max})n \log n)$ where $d_{\max}$ is the latest deadline. Note that in any feasible instance we have $d_{\max} \geq p \left\lceil \frac{n}{m} \right\rceil$. Vakhania's algorithm is therefore competitive when the processing time p is small and the number of machines is proportional to the number of jobs. Dürr and Hurand [3] gave an algorithm that runs in $O(n^3)$. Even though their algorithm does not improve over the best time complexity, it deepens the understanding of the problem by reducing it to a shortest path in a graph. The algorithm presented in this paper is based on this reduction while improving substantially its time complexity.

There exist specializations to the problem with more efficient algorithms. For instance, on a single machine ($m = 1$) and with a unit processing time ($p = 1$), the problem consists of a matching in a convex bipartite graph. Lipski and Preparata [7] designed an algorithm running in $O(n\alpha(n))$ time where α is the inverse of Ackermann's function and where it is assumed that the jobs are presorted by deadlines. Gabow and Tarjan's [4] showed how to reduce this complexity to $O(n)$ using their union-find data structure. The algorithm can be easily adapted without altering the complexity for multiple machines even in the case where the number of machines fluctuates over time. Finally, Garey et al. [5] solve the scheduling problem in $O(n \log n)$ time for jobs of equal processing times on a single machine ($m = 1$).

2 Reduction to a Shortest Path

Suppose that an oracle provides the number x_t of jobs starting at time t. A solution can be constructed using a matching in a convex bipartite graph. For each job j, we create a node j and for each time t, we create x_t duplicates of a node t. There is an edge between a job node j and a time node t if $r_j \leq t < u_j$. A matching in such a graph associates to each job a starting time. Since the graph is convex, Lipski and Preparata [7] show how to find such a matching in $O(n\alpha(n))$ time.

Thus we have reduced the scheduling problem to finding how many jobs start at any given time t. We answer this question by computing a shortest path in a graph in a manner similar to what Dürr and Hurand [3] did. Their solution consists of building a graph with $O(n)$ nodes and $O(n^2)$ edges and to compute the shortest path using the Bellman-Ford algorithm in $O(n^3)$ time. We propose a similar approach with a graph having more nodes. These additional nodes make the computation of a solution easier. However, the main contribution of our technique is presented in Section 3 to 5 where we identify and exploit the properties of the graph to obtain a substantially faster algorithm.

We know that at most m jobs can start in any window of size p. We can already state the equations. Let $r_{\min} = \min_i r_i$ and $u_{\max} = \max_i u_i$ be the earliest release time and latest allowed starting time.

$$\sum_{j=t}^{t+p-1} x_j \leq m \qquad \forall\, r_{\min} \leq t \leq u_{\max} - p \tag{4}$$

$$x_t \geq 0 \qquad \forall\, r_{\min} \leq t < u_{\max} \tag{5}$$

Condition 4 states that at most m processes may start on a given interval of length p. Let $u_{\max} = \max_i u_i$ be the latest time when a job can start. Given two (possibly identical) jobs i and j defining a non-empty semi-open interval $[r_i, u_j)$, the set $\{k \mid r_i \leq r_k \wedge u_k \leq u_j\}$ denotes the jobs that must start in this interval, hence

$$\sum_{t=r_i}^{u_j-1} x_t \geq |\{k \mid r_i \leq r_k \wedge u_k \leq u_j\}| \qquad \forall\, i, j \in 1..n \tag{6}$$

▶ **Lemma 1.** *The scheduling problem has a solution if and only if Equations* (4) *to* (6) *are satisfiable.*

Proof. ($\Rightarrow$) Given a valid schedule, we set x_t to the number of jobs starting at time t, i.e $x_t = \{i \mid s_i = t\}$. By definition of the problem, all equations are satisfied.
($\Leftarrow$) Consider a bipartite graph $G = \langle J \cup T, E\rangle$ such that $J = \{1, \ldots, n\}$ are the nodes associated to the jobs and T is a multiset of time points such that time t occurs x_t times in T. There is an edge from the job-node i to the time-node t if $t \in [r_i, u_i]$. Note that the bipartite graph is convex, i.e. if there is an edge (i, t_1) and an edge (i, t_3) then there is an edge (i, t_2) for all $t_2 \in [t_1, t_3] \cap T$. A maximum matching in this convex bipartite graph gives a valid assignment of jobs to time points. Equation (4) ensures that no machines are overloaded and the schedule is feasible. From Hall's [6] marriage theorem, there exists a matching if and only if for any set of jobs S, there are at least $|S|$ time nodes that are adjacent to the nodes in S. Let $i \in S$ be the job with the earliest release time and $j \in S$ be the job with the latest deadline. Inequality (6) ensures that there are at least $|\{k \mid r_i \leq r_k \wedge u_k \leq u_j\}| \geq |S|$ time-nodes adjacent to the nodes in S which meets Hall's condidtion. ◀

We perform a change of variables to simplify the form of the equations. Let $y_t = \sum_{i=r_{\min}}^{t-1} x_i$ for $r_{\min} \leq t \leq u_{\max}$. Equations (4) to (6) are rewritten as follows.

$$y_{t+p} - y_t \leq m \qquad \forall\, r_{\min} \leq t \leq u_{\max} - p \tag{7}$$

$$y_t - y_{t+1} \leq 0 \qquad \forall\, r_{\min} \leq t < u_{\max} \tag{8}$$

$$y_{r_i} - y_{u_j} \leq -|\{k \mid r_i \leq r_k \wedge u_k \leq u_j\}| \qquad \forall\, r_i < u_j \tag{9}$$

Equations (7) to (9) form a system of difference constraints, which can be solved creating a graph with one node per variable and an edge (a, b) of weight w for each constraint $b - a \leq w$. For the equations above, we obtain a graph $G = \langle T, E\rangle$ where $T = r_{\min}..u_{\max}$ is the set of nodes, one for each integer time point. We consider three sets of edges: *forward edges* $E_f = \{(t, t+p) \mid r_{\min} \leq t \leq u_{\max} - p\}$, *backward edges* $E_b = \{(u_j, r_i) \mid r_i < u_j\}$, and *null edges* $E_n = \{(t+1, t) \mid r_{\min} \leq t < u_{\max}\}$. The edges of the graph are the union of these three sets of edges $E = E_f \cup E_n \cup E_b$ that are directly derived from Equations (7), (8), and (9). The following weight function maps every edge to an integer weight. Let $(a, b) \in E$, then

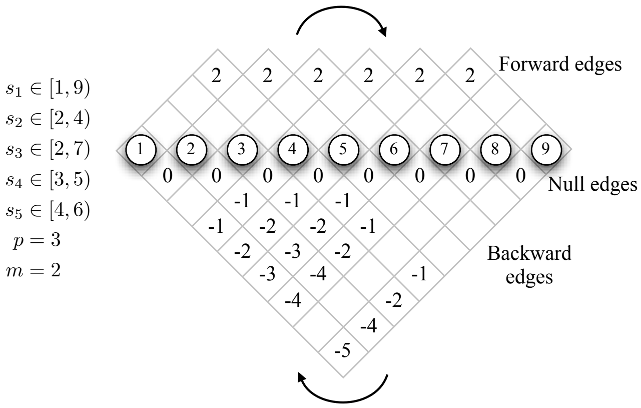
$$w(a,b) = \begin{cases} m & \text{if } a < b \\ -|\{k \mid b \leq r_k \wedge u_k \leq a\}| & \text{otherwise} \end{cases} \tag{10}$$

We call the graph G the *scheduling graph.* The following theorem shows the connexion between a shortest path in the scheduling graph and a solution to the system of difference constraints. The proof is taken from Cormen et al. [2] who credit it to R. Bellman.

▶ **Theorem 2.** *Let $\delta(a, b)$ be the shortest distance between node a and node b in the scheduling graph. The assignment $y_t = n + \delta(u_{\max}, t)$ is a solution to Equations* (7) *to* (9)*.*

Proof. Suppose there is an inequality $y_b - y_a \leq w(a, b)$ that is not satisfied by the assignment, we therefore have $n + \delta(u_{\max}, b) - n - \delta(u_{\max}, a) > w(a, b)$. The inequality $\delta(u_{\max}, b) > \delta(u_{\max}, a) + w(a, b)$ contradicts that $\delta(u_{\max}, b)$ is the shortest distance from $u_{\max}$ to b. ◀

Let $|T| \in O(u_{\max} - r_{\min})$ be the number of nodes and $|E| \in O(n^2 + u_{\max} - r_{\min})$ the number of edges in the scheduling graph. Here we could directly apply a shortest path algorithm such as Bellman-Ford to compute a shortest path from $u_{\max}$ to all other nodes in the graph. These algorithms run in time polynomial in T and $|E|$, or in other words,



i	$\delta(9, i)$
1	-5
2	-5
3	-4
4	-3
5	-3
6	-2
7	-1
8	-1
9	0

Figure 1 A scheduling graph with 9 nodes. The weight on an edge between two nodes appear at the intersection of the two diagonals passing by these nodes. The weights of the null and backward edges appear bellow the nodes while the weights of the forward edges appear above the nodes. Empty cells indicate the absence of an edge. For instance, the weight of backward edge $(9, 2)$ is -4. The shortest path between node 9 and 6 passes by the nodes 9, 1, 4, 7, 2, 5, 3, 6.

pseudo-polynomial on the term $u_{\max}$. In the next section we show properties of the scheduling graph and use them to propose a much more efficient method based on a speed up version of Bellman- Ford's algorithm.

Figure 1 presents a scheduling graph with 2 machines, 5 jobs, and a processing time of 3.

3 Properties of the Scheduling Graph

▶ **Lemma 3.** *Let $e < f < g < h$ be four nodes in a scheduling graph without negative cycles. If the edges (h, f) and (g, e) lie on a shortest path then there exists an equivalent path of same length that does not include these edges.*

Proof. Suppose that the edges (h, f) and (g, e) lie on a same shortest path and that (h, f) precedes (g, e) on this path. Since any sub-path of a shortest path is also a shortest path, we have

$$w(h, e) \geq w(h, f) + \delta(f, g) + w(g, e) \tag{11}$$

Recalling that $-w(y, x)$ for $x < y$ is the number of jobs that must start in the time interval $[x, y)$, we know that the following relationship holds on backward edges.

$$w(h, e) \leq w(g, e) + w(h, f) - w(g, f) \tag{12}$$

Subtracting (12) from (11) shows that the cycle passing by (f, g) is negative or null and since there are no negative cycles, we obtain the equality $0 = w(g, f) + \delta(f, g)$ Substituting this equality back in (12) shows an equality in (11). The edge (h, e) is therefore an equivalent path that does not contain the edges (h, f) nor (g, e).

Alternatively, suppose that (h, f) succeeds to (g, e) on the path. We have

$$w(g, f) \geq w(g, e) + \delta(e, h) + w(h, f) \tag{13}$$

Adding (12) to (13) gives $0 \geq w(h, e) + \delta(e, h)$. Since there are no negative cycles in the scheduling graph, the inequality is tight. Substituting the equality into (12) shows that (13) is tight and that (g, f) is an equivalent shortest path. ◀

A backward edge (b, a) is associated to the interval $[a, b)$. If two backward edges have disjoint intervals, we say that the backward edges are *disjoint*. If the interval of one backward edge is contained in the interval of another backward edge, we say that the backward edges are *nested*. Otherwise, we say that the backward edges are *crossed*.

▶ **Lemma 4.** *The shortest path which also minimizes the number of edges does not have crossed backward edges.*

Proof. By applying Lemma 3 on a shortest path, one obtains a shortest path with two crossed edges and one forward edge replaced by one backward edge. One can repeat the process until there are no more crossed edges in the path. Since each time we apply Lemma 3, the number of edges in the path diminishes, the process is guaranteed to finish. ◀

Let d be the distance vector such that $d[t] = \delta(u_{\max}, t)$ is the shortest distance from node $u_{\max}$ to node t. The vector d is monotonically increasing.

▶ **Lemma 5.** *The distance vector d is monotonically increasing.*

Proof. Consider the nodes t and $t + 1$, the null edges E_n guarantees that $d[t] \leq d[t + 1] + w(t + 1, t)$ or simply $d[t] \leq d[t + 1]$. ◀

▶ **Lemma 6.** *If the scheduling graph has no negative cycles, $d[r_{\min}] = -n$.*

Proof. Lemma 4 implies that there is a shortest path from $u_{\max}$ to $r_{\min}$ that do not have forward edges. Because two consecutive backward edges are no shorter than one longer backward edge ($w(c, a) \leq w(c, b) + w(b, a)$ for any time point $a < b < c$) we conclude that the edge $(u_{\max}, r_{\min})$ is a single-segment shortest path from $u_{\max}$ to $r_{\min}$ with distance $-n$. ◀

Lemma 5 and Lemma 6 implies that the distance vector d contains the values $-n..0$ in non-decreasing order. Keeping a structure in memory of the n time points where the vector d is strictly increasing is sufficient to retrieve all components of vector d. This is a first step towards a strongly polynomial algorithm.

4 The Algorithm

We present an algorithm based on the Bellman-Ford algorithm [1] for the single-source shortest path problem. We encode the distance vector with a vector d^{-1} of dimension $n + 1$. The component $d^{-1}[i]$ is the rightmost node reachable at distance at most $-i$. For example, if $n = 10$ and $d^{-1}[3] = 4$, there is a path from $u_{\max}$ to 4 of weight at most -3.

An iteration of the Bellman-Ford algorithm applies the relaxation $d[b] \leftarrow \min_{(a,b) \in E} d[a] + w(a, b)$ for all nodes b and assumes that there is an edge (b, b) of null weight on all nodes. After sufficiently many iterations. the algorithm converges to a distance vector d such that $d[a]$ is the shortest distance between the source node and the node a.

We develop two procedures. One that applies the relaxation to the edges in $E_n \cup E_f$ and one that applies it to the edges in $E_n \cup E_b$. Yen [12] introduced the technique of partitioning edges between forward and backward edges to reduce the number of iterations of the Bellman-Ford algorithm to the number of times a shortest path alternates between a backward edge to a forward edge. In a scheduling graph, the number of alternations can be reduced to $\min(n, \lceil \frac{n}{m} \rceil p)$ as we will prove in Section 5. The algorithm for finding the starting times adapts the Bellman-Ford algorithm to the scheduling graph. If the distance vector of the algorithm does not converge after a sufficient number of iterations there exists a negative cycle in the graph proving that the problem is unsolvable. The algorithm then returns an error message.

```
Algorithm 1: FindStartingTimes($\vec{r}, \vec{u}, m, p$)
    $B \leftarrow sort(\{r_i \mid i \in 1..n\} \cup \{u_i \mid i \in 1..n\})$;
    for $i = 1$ to $n$ do $l_i \leftarrow index(B, r_i)$;
    for $i = 1$ to $n$ do $v_i \leftarrow index(B, u_i)$;
    $d_0 \leftarrow [u_{\max}, \underbrace{r_{\min}, \dots, r_{\min}}_{n \text{ copies}}]$;
    for $k \leftarrow 1$ to $\min(n, \lceil \frac{n}{m} \rceil p) + 1$ do
        $\vec{d_k} \leftarrow RelaxForwardEdges(\vec{d_{k-1}}, m, p)$;
        $\vec{d_k} \leftarrow RelaxBackwardEdges(\vec{d_k}, \vec{l}, \vec{v}, B)$;
        if $\vec{d_k} = \vec{d_{k-1}}$ then return $[d_k[n-1], d_k[n-2], \dots, d_k[0]]$;
    return Failure;
```

Relaxing the Forward Edges.

Relaxing forward edges is done in $O(n)$ time by iterating over the distance vector d^{-1}. It ensures that if there is a path of distance i that goes to node x, then there is a path of distance $i + m$ that reaches node $x + p$. For all possible distances i spanning from $-n$ to $-m$, we apply the relaxation $d^{-1}[-i - m] \leftarrow \max(d^{-1}[-i] + p, d^{-1}[-i - m])$.

```
Algorithm 2: RelaxForwardEdges($\vec{d}, m, p$)
    for $i \leftarrow -n$ to $-m$ do
        $d[-i - m] \leftarrow \max(d[-i] + p, d[-i - m])$;
    return $\vec{d}$;
```

Relaxing the Backward Edges.

Processing backward edges in $O(n)$ time is more complex. Assume that jobs are sorted in non-decreasing order of release times ($r_i \leq r_{i+1}$). The algorithm is based on the similarity between the backward edges incoming to node r_i and backward edges incoming to node r_{i+1}.

▶ **Lemma 7.** *Let J_i be the set of jobs sharing the same release time as r_i. The backward edges incoming to r_i and r_{i+1} are linked by the relation $w(t, r_i) = w(t, r_{i+1}) - |\{k \in J_i \mid u_k \leq t\}|$.*

Proof. Recall that $w(a, b)$ is the negation of the number of jobs that must start in the interval (b, a). The number of jobs that must start in the time interval $[r_i, t)$ is the number of jobs that *must* start in the interval $[r_{i+1}, t)$ plus the number of jobs that *can* start in $[r_i, r_{i+1})$ but must start before t. Hence $-w(t, r_i) = -w(t, r_{i+1}) + |\{k \in J_i | u_k \leq t\}|$ as claimed. ◀

Let $d[u_j]$ be the best distance found so far by the Bellman-Ford algorithm from node $u_{\max}$ to node u_j. Relaxing the backward edges consists of computing the value $\min_j d[u_j] + w(u_j, r_i)$ for all r_i. The following Lemma shows that not all edges need to be processed.

▶ **Lemma 8.** *Given two latest starting times $u_a < u_b$, if $d[u_a] + w(u_a, r_{i+1}) \geq d[u_b] + w(u_b, r_{i+1})$ then $d[u_a] + w(u_a, r_i) \geq d[u_b] + w(u_b, r_i)$.*

Proof. Using the set J_i as defined in Lemma 7, we have $|\{j \in J_i \mid u_j \leq u_a\}| \leq |\{j \in J_i \mid u_j \leq u_b\}|$. From Lemma 7, we obtain $w(u_a, r_{i+1}) - w(u_a, r_i) \leq w(u_b, r_{i+1}) - w(u_b, r_i)$. Subtracting this inequality from the hypothesis proves the Lemma. ◀

Lemma 8 is fundamental to obtain a fast algorithm relaxing the backward edges. It says that when processing the backward edges ingoing to r_{i+1}, if the edge (u_b, r_{i+1}) is a better or equivalent candidate for a shortest path than (u_a, r_{i+1}) then the edge (u_b, r_i) is also a better or equivalent candidate than (u_a, r_i). By transitivity, any backward edge outgoing from u_a and ingoing to a node smaller than r_{i+1} can be ignored.

Let B be the set containing all the release times $r_1, \ldots, r_n$ and latest starting times $u_1, \ldots, u_n$. This set contains no duplicates and its elements are labeled from b_1 to $b_{|B|}$.

We construct a singly linked-list that we call the *list of representatives.* The list initially contains the elements of B in increasing order. Each element of the list is a *representative* of a set that initially only contains itself. The representative is always the largest element of its set. Each set is represented in the data structure by a node labeled by its representative that has a link to the previous node $b_1 \leftarrow b_2 \leftarrow \cdots$. The link between b_{j+1} and b_j has weight $d[b_{j+1}] - d[b_j]$. If the weight of a link (b_{j+1}, b_j) is null, we merge the node b_j and the node b_{j+1} together to form a larger set for which b_{j+1} is the representative. The node b_j disappears from the list of representatives since Lemma 8 ensures that b_{j+1} will always be a better candidate than b_j.

On the running example of Figure 1, the vector $\vec{d}$ is initialized to $\vec{d} = [9, 1, 1, 1, 1, 1]$. After the forward edge relaxation stage, its value becomes $\vec{d} = [9, 7, 4, 4, 1, 1]$. The representatives are $B = \{1, 2, 3, 4, 5, 6, 7, 9\}$ which gives the vector $d = [-5, -3, -3, -3, -1, -1, -1, 0]$ that maps each element in B to a distance. After merging the sets connected with a null link, we obtain the following chain where representatives are highlighted in bold.

$$\{\mathbf{1}\} \xleftarrow{2} \{2, 3, \mathbf{4}\} \xleftarrow{2} \{5, 6, \mathbf{7}\} \xleftarrow{1} \{\mathbf{9}\} \tag{14}$$

Initially, the data structure allows us to compute $d[b_j]$ for any j. As we shall show in Lemma 9, one only needs to visit the nodes from b_j to b_1 and sum up the weights on the links to obtain $d[b_j] + n$. Subtracting n from the sum of the links gives $d[b_j]$. The data structure can be updated to compute the values $d[b_j] + w(b_j, r_i)$ for each backward edge (b_j, r_i) in the scheduling graph. We process the tasks in non-increasing order of release time starting with r_n. When processing task i, we first look in the data structure for the representative of u_i which we call b_q. Assume that the node b_q points to b_t, we decrement the weight of the link (b_q, b_t). If the weight of the link becomes null after decrementing, we delete b_t from the list of representatives and merge the set containing b_t with the set represented by b_q. The element b_q remains the representative of the merged set.

Continuing with the running example, processing job 5 decreases by one the weight on the link between node 7 and node 4. Processing job 4 decreases once more the weight of this link and fix it to zero. The data structure then looks as follows.

$$\{\mathbf{1}\} \xleftarrow{2} \{2, 3, 4, 5, 6, \mathbf{7}\} \xleftarrow{1} \{\mathbf{9}\} \tag{15}$$

▶ **Lemma 9.** *After processing the last job with release time r_i, the sum of the weights on the links from the representative of u_j to node b_1 is equal to $d[u_j] + w(u_j, r_i) + n$.*

Proof. Let b_k be the representative of u_j. Initially, the weights on the links of the data structure from b_k to b_1 are equal to the telescopic sum $\sum_{l=0}^{k-1} (d[b_{l+1}] - d[b_l])$ which simplifies to $d[b_j] - d[b_1] = d[b_j] + n$. After processing the last job with release time r_i, all jobs that must start at or after r_i and before b_k have been processed. Each of these $|\{a \mid r_i \leq r_a \wedge u_a \leq u_j\}|$ jobs decremented by one a link on the path between b_k and b_1. Therefore, the sum of the links on a path between u_j and b_1 is $d[u_j] + w(u_j, r_i) + n$. ◀

The Bellman-Ford algorithm requires to find the backward edge incoming into r_i that minimizes the value $d'[b_j] + w(b_j, r_i)$ where b_j can be r_i. Lemma 10 shows how the data structure finds the optimal edge.

▶ **Lemma 10.** *Let b_j be the representative of r_i after processing all jobs with release times greater than or equal to r_i. The backward edge (b_j, r_i) is the one minimizing the value $d'[b_j] + w(b_j, r_i)$.*

Proof. The backward edge (b_j, r_i) that minimizes $d'[b_j] + w(b_j, r_i)$ also minimizes $d'[b_j] + w(b_j, r_i) + n$. Lemma 9 guarantees that the later value is equal to the weights on the path from b_j to b_1. Since all weights on the path are positive, the smallest representative that is greater than or equal to r_i is the one minimizing $d'[b_j] + w(b_j, r_i)$. This representative is necessarily the representative of r_i.

We prove that values that are not representatives are not optimal. If u_b is the representative of u_a, then for some release time $r_c \geq r_i$, the link from node r_b to r_a was decremented to zero. From Lemma 9, we have $d'[u_a] + w(u_a, r_c) = d'[u_b] + w(u_b, r_c)$. From Lemma 8 we conclude that $d'[u_a] + w(u_a, r_i) \geq d'[u_b] + w(u_b, r_i)$. Therefore, the representative u_b is as good or better than the non-representative u_a. ◀

The algorithm RelaxBackwardEdges uses the data structure discussed above to relax the backward edges. The first for loop on line 1 converts the vector d to the vector d'. Recall that $d[i]$ is the largest node in the graph reachable at distance at most $-i$ and $d'[i]$ is the smallest distance found so far to reach node $B[i]$ where B is the sorted vector of release times r_i and latest starting times u_i.

The algorithm then initializes the data structure. Each node is a set in a union-find data structure T whose representative is the largest element. The weight of the link pointing to a representative b_i is stored in $c[i]$. We store in $k[i]$ the number of jobs j that have been processed so far and for which the latest starting time u_j is represented by b_i. The for loop on line 2 processes each job in non-increasing order of release time. The data structure is updated as explained above. Line 3 computes the value $d'[b_e] + w(b_e, r_i)$ where b_e is the representative of r_i. Note that $k[e]$ is the number of processed jobs with latest starting time smaller than or equal to b_e which is equal to $-w(b_e, r_i)$.

5 Analysis

The following lemmas show the correctness of the algorithm and give the conditions to bound its time complexity.

▶ **Lemma 11.** *There is a shortest path from $u_{\max}$ to all other nodes with at most $\lceil \frac{n}{m} \rceil$ disjoint backward edges.*

Proof. Suppose a shortest path has k disjoint edges (b_i, a_i) for $1 \leq i \leq k$. We assume that these backward edges are interleaved with forward edges since two backward edges connected with null edges can be replaced by a single backward edge of cost equal or smaller than the sum of the weights of the backward edges. Since the intervals $[a_i, b_i)$ are disjoint, we have $\sum_{i=1}^{k} w(b_i, a_i) \geq -n$. It is safe to assume that the path has negative weight since a path of null weight can be entirely constituted of null edges without any backward edges. The path has $k - 1$ sequences of forward edges whose weights sum to at most $n - 1$. Each sequence of forward edges must be at least of weight m. To maximize the number k, we assume that each sequence of forward edges has a single edge. We have $(k-1)m \leq n - 1$ which implies $k \leq \lfloor \frac{n-1}{m} \rfloor + 1 = \lceil \frac{n}{m} \rceil$. ◀

Algorithm 3: RelaxBackwardEdges($\vec{d}, \vec{l}, \vec{v}, B, W$)

```
  Construct a vector $d'$ s.t. the distance between $u_{\max}$ and $B[i]$ is $d'[i]$;
  $d' \leftarrow [\,]$;                                         // Empty vector
  $j \leftarrow -n$;
1 for $b \in B$ in increasing order do
      while $b > d[-j]$ do $j \leftarrow j + 1$;
      append($d', j$);
  $T \leftarrow UnionFind(|B|)$ ;                               // $|B|$ disjoint sets
  $k \leftarrow [0, \dots, 0]$;                 // Create a null vector of dimension $|B|$
  for $i \leftarrow 1$ to $|B| - 1$ do
      $c[i] \leftarrow d'[i+1] - d'[i]$;
      if $c[i] = 0$ then $Union(T, i, i+1)$;
2 for $i \in [1, n]$ in non-increasing value of $l_i$ do
      $q \leftarrow FindMax(T, v_i)$;
      $t \leftarrow FindMax(T, FindMin(T, v_i) - 1)$;
      $c[t] \leftarrow c[t] - 1$;
      $k[q] \leftarrow k[q] + 1$;
      if $c[t] = 0$ then
          $Union(T, t, q)$;
          $k[q] \leftarrow k[q] + k[t]$;
      $e \leftarrow FindMax(T, l_i)$;
3     $a \leftarrow d'[e] - k[e]$;
      if $a < d'[l_i]$ then
          $d'[l_i] \leftarrow a$;
          $d[-a] \leftarrow B[l_i]$;
  return $\vec{d}$;
```

Lemma 12 gives an upper bound on the number of nested backward edges lying on a shortest path. Lemma 13 gives an upper bound on the number of backward edges lying on a shortest path.

▶ **Lemma 12.** *A shortest path can have at most p nested backward edges (b_i, a_i) such that $a_i < a_{i+1}$ and $b_i > b_{i+1}$.*

Proof. In such a path, there must be a sequence of forward edges that connects a_i to b_j for some i and some j. This sequence of forward edges cannot pass by a node a_k for $k > i$ since each node is visited only once on a path. This implies that a_i and a_k are not congruent modulo p for every $k > i$. Consequently, $a_i \neq a_j \mod p$ for all $i \neq j$. The maximum set of values satisfying this property has cardinality p. ◀

▶ **Lemma 13.** *There is a shortest path with at most $\min(n, \lceil \frac{n}{m} \rceil p)$ backward edges.*

Proof. The number of backward edges on a path is bounded by n since there are at most n nodes r_i to which they lead to. Lemma 11 guarantees that there are at most $\lceil \frac{n}{m} \rceil$ disjoint backward edges. Each disjoint backward edge can have at most p nested backward edges. Therefore, there are at most $\lceil \frac{n}{m} \rceil p$ backward edges on a shortest path. The number of backward edges is bounded by the smallest of both bounds. ◀

▶ **Theorem 14.** *The algorithm for finding the starting times is correct.*

Proof. The correctness of the forward and backward edge relaxation algorithms follows from the discussions and Lemmas in the previous section. The correctness of algorithm for finding the starting times follows from the Bellman-Ford algorithm. We however need to justify why $\min(n, \lceil \frac{n}{m} \rceil p)$ iterations are sufficient. A path is necessarily an alternation of forward edges in E_f and backward or null edges in $E_b \cup E_n$. Yen [12] shows that the number of iterations can be bounded to the number of alternations. An alternation between a forward edge, a sequence of null edges, and another forward edge can be replaced by an equivalent path of two forward edges and a sequence of null edges (or a sequence of null edges followed by two forward edges). Consequently, we can assume that the sequences of null edges occur before or after a backward edge. Lemma 13 gives an upper bound of $\min(n, \lceil \frac{n}{m} \rceil p)$ on the number of backward edges which is also an upper bound on the number of alternations. ◀

▶ **Theorem 15.** *The algorithm for finding the starting times completes in $O(\min(1, \frac{p}{m})n^2)$ steps.*

Proof. The running time complexity of the forward edge relaxation stage is clearly $O(n)$. The complexity of the backward edge relaxation stage depends on the implementation of the Union-Find data structure. There are $O(n)$ calls to the functions $FindMin$, $FindMax$, and $Union$. Using path compression, each call can be executed in $O(\alpha(n))$ time where α is the inverse of Ackermann's function. However, since the disjoint sets always contain consecutive values in B, Gabow and Tarjan [4] propose a data structure where each call executes in constant amortized time which makes the backward edge relaxation stage run in $O(n)$ steps. Finally, the algorithm for finding the starting times performs $min(n, \lceil \frac{n}{m} \rceil p)$ calls to the forward and backward edge relaxation stages which results in a running time complexity of $O(\min(1, \frac{p}{m})n^2)$. ◀

Observe that the running time complexity $O(\min(1, \frac{p}{m})n^2)$ is strongly polynomial in all parameters. While the presence of the variable p might suggest pseudo-polynomiality, the term $\min(1, \frac{p}{m})$ is always bounded by a constant. In the particular case where p is considered to be a small bounded value, the resulting complexity $O(\frac{n^2}{m})$ decreases as the number of machines m increases.

The algorithm has better performance in some special cases of interest. For instance, when $p = 1$, a shortest path cannot have nested backward edges as stated in Lemma 12. Neither can the path have disjoint backward edges. To wit, suppose that (a, b) and (c, d) are two disjoint backward edges such that (a, b) occurs before (c, d) on a shortest path. If $b \geq c$ then the edge (d, a) is an equivalent or shorter path. If $b < c$ then the forward edges need to pass by node c before reaching d creating a loop. With shortest paths including only one backward edge, the algorithm converges after one iteration.

We showed that the algorithm computes in polynomial time a shortest path and thus a valid schedule. We show that it also detects infeasibility in polynomial time. By Lemma 13, if there is no negative cycle, a shortest path has at most $\min(n, \lceil \frac{n}{m} \rceil p)$ alternations between backward and forward edges. By Yen's theorem, if there are no negative cycles, the Bellman-Ford algorithm will converge after $\min(n, \lceil \frac{n}{m} \rceil p)$ iterations. On the other hand, if the graph has a negative cycle, the Bellman-Ford never converges since the cost function tends to minus infinity. To detect infeasibility, the algorithm first iterates $\min(n, \lceil \frac{n}{m} \rceil p)$ times. If the graph has no negative cycles, the algorithm should have converged. To test if it did converge, the algorithm iterates one more time. If the distance vector has changed, then the algorithm has not converged and will never do. The algorithm detected infeasibility in exactly $\min(n, \lceil \frac{n}{m} \rceil p) + 1$ iterations which is strongly polynomial.

We now characterize the solution returned by the algorithm and prove that it minimizes both the sum of the completion times and the makespan.

▶ **Theorem 16.** *The solution returned by the algorithm for finding the starting times minimizes the sum of the completion times.*

Proof. Minimizing the sum of the completion times is equivalent to minimizing the sum of the starting times. We recall that x_t is the number of jobs starting at time t. Minimizing the sum of the starting times is equivalent to minimizing $\sum_{t=r_{\min}}^{u_{\max}-1} tx_t$. Performing the change of variables from x_t to $y_{t+1} - y_t$ leads to a telescopic sum that we solve in (16) and simplify in (17).

$$\sum_{t=r_{\min}}^{u_{\max}-1} t(y_{t+1} - y_t) = (u_{\max} - 1)y_{u_{\max}} - r_{\min}y_{r_{\min}} - \sum_{t=r_{\min}+1}^{u_{\max}-1} y_t \tag{16}$$

$$= r_{\min}(y_{u_{\max}} - y_{r_{\min}}) + \sum_{t=r_{\min}+1}^{u_{\max}-1} (y_{u_{\max}} - y_t) \tag{17}$$

The difference $y_{u_{\max}} - y_{r_{\min}}$ is equal to n for all solutions since this is the number of jobs executed between the beginning and the end of the schedule. The first term to optimize is therefore a constant and can be ignored leaving only the expression $\sum_{t=r_{\min}+1}^{u_{\max}-1}(y_{u_{\max}} - y_t)$ to minimize or $\sum_{t=r_{\min}+1}^{u_{\max}-1}(y_t - y_{u_{\max}})$ to maximize.

Let $(a_1, a_2), (a_2, a_3), \ldots, (a_{k-1}, a_k)$ with $a_1 = u_{\max}$ and $a_k = t$ be the edges on the shortest path from $u_{\max}$ to t with total weight $\delta(u_{\max}, t)$. By substituting the inequalities (7)-(9), we obtain this relation.

$$\delta(u_{\max}, t) = \sum_{i=1}^{k-1} w(a_i, a_{i+1}) \geq \sum_{i=1}^{k-1} y_{a_{i+1}} - y_{a_i} = y_t - y_{u_{\max}} \tag{18}$$

By setting $y_{u_{\max}} = 0$ and $y_t = \delta(u_{\max}, t)$, we maximize the difference $y_t - y_{u_{\max}}$ up to reaching the equality. Since we maximize the difference for all values t, this maximizes $\sum_{t=r_{\min}}^{u_{\max}-1}(y_t - y_{u_{\max}})$ which is equivalent to minimizing $\sum_{t=r_{\min}}^{u_{\max}-1} tx_t$. ◀

▶ **Theorem 17.** *The algorithm for finding the starting times minimizes the makespan.*

Proof. Suppose the algorithm makes the latest job start at time m. We have $-\sum_{t=m}^{u_{\max}-1} x_t = y_m - y_{u_{\max}} < 0$. A schedule with smaller makespan would have $y_m - y_{u_{\max}} = 0$ which means this quantity would be greater than the one produced by the algorithm. However, following the argument in Theorem 16, the inequality (18) is maximized up to equality. Therefore, a schedule with a smaller makespan violates (7)-(9). ◀

6 Conclusion

We gave an algorithm which substantially improves over the previous best known ones for the problem of makespan and completion times minimization for multi-machine scheduling with tasks of equal length. We observed that the running time complexity depends on the relative sizes of m and p. An open question is to show whether this relationship is tight or whether there exists a better complexity in terms of n, m, and p.

References

1 R. E. Bellman. On a routing problem. *Quarterly Applied Mathematics*, 16:87–90, 1958.

2 T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. S. Stein. *Introduction to Algorithms*. MIT Press, 2001.

3 C. Dürr and M. Hurand. Finding total unimodularity in optimization problems solved by linear programs. *Algorithmica*, 2009. Published online on April 22nd, 2009, DOI 10.1007/s00453-009-9310-7.

4 H. N. Gabow and R. E. Tarjan. A linear-time algorithm for a special case of disjoint set union. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 246–251, 1983.

5 M.R. Garey, D.S. Johnson, B.B. Simons, and R.E. Tarjan. Scheduling unit-time tasks with arbitrary release times and deadlines. *SIAM Journal on Computing*, 10(2):256–269, 1981.

6 P. Hall. On representatives of subsets. *Journal of the London Mathematical Society*, pages 26–30, 1935.

7 W. Lipski and F. P. Preparata. Efficient algorithms for finding maximum matchings in convex bipartite graphs and related problems. *Acta Informatica*, 15:329–346, 1981.

8 Michael Pinedo. *Scheduling: theory, algorithms, and systems*. Springer, third edition, 2008.

9 B. Simons. Multiprocessor scheduling of unit-time jobs with arbitrary release times and deadlines. *SIAM Journal of Computing*, 12(2):294–299, May 1983.

10 B. Simons and M. K. Warmuth. A fast algorithm for multiprocessor scheduling of unit-length jobs. *SIAM Journal of Computing*, 18(4):690–710, August 1989.

11 N. Vakhania. Scheduling equal-length jobs with delivery times on identical processors. *International Journal of Computer Mathematics*, 79(6):715–728, 2002.

12 J. Y. Yen. An algorithm for finding shortest routes from all source nodes to a given destination in general network. *Quarterly Applied Mathematics*, 27:526–530, 1970.

Scheduling for Weighted Flow Time and Energy with Rejection Penalty*

Sze-Hang Chan[1], Tak-Wah Lam[2], and Lap-Kei Lee[3]

1,2 Department of Computer Science, University of Hong Kong, Hong Kong.
{shchan, twlam}@cs.hku.hk
3 Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany.
lklee@mpi-inf.mpg.de

Abstract

This paper revisits the online problem of flow-time scheduling on a single processor when jobs can be rejected at some penalty [4]. The *user cost* of a job is defined as the weighted flow time of the job plus the penalty if it is rejected before completion. For jobs with arbitrary weights and arbitrary penalties, Bansal et al. [4] gave an online algorithm that is $O((\log W + \log C)^2)$-competitive for minimizing the total user cost when using a slightly faster processor, where W and C are the max-min ratios of job weights and job penalties, respectively. In this paper we improve this result with a new algorithm that can achieve a constant competitive ratio independent of W and C when using a slightly faster processor. Note that the above results assume a processor running at a fixed speed. This paper shows more interesting results on extending the above study to the dynamic speed scaling model, where the processor can vary the speed dynamically and the rate of energy consumption is a cubic or any increasing function of speed. A scheduling algorithm has to control job admission and determine the order and speed of job execution. This paper studies the tradeoff between the above-mentioned user cost and energy, and it shows two O(1)-competitive algorithms and a lower bound result on minimizing the user cost plus energy. These algorithms can also be regarded as a generalization of the recent work on minimizing flow time plus energy when all jobs must be completed (see the survey paper [1]).

1998 ACM Subject Classification F.2.2[Analysis of Algorithms and Problem Complexity] Non-numerical Algorithms and Problems—Sequencing and scheduling

Keywords and phrases Online scheduling, weighted flow time, rejection penalty, speed scaling

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.392

1 Introduction

It is not uncommon that a server rejects some jobs (in particular, low-priority jobs) during peak load, yet it is non-trivial how to strike a balance between the cost due to longer response time and the cost of rejecting some jobs. Bansal et al. [4] initiated the study of flow-time scheduling on a single processor when jobs can be rejected at some penalty. Specifically, jobs are released online with arbitrary sizes, weights and penalties. Consider a schedule which may reject some jobs before completion, each job defines a *user cost* equal to its weighted flow time plus the penalty if it is rejected, where the flow time is the time elapsed since a job is released until it is completed or rejected. In this penalty model, the scheduler aims at minimizing the total user cost of all jobs.

* This research was mainly done when the first two authors were visiting MPI, whose hospitality was greatly appreciated.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 392–403
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

Assuming jobs have uniform penalty and unit weight, Bansal et al. [4] gave an online algorithm that is 2-competitive for minimizing the total user cost. For jobs with arbitrary penalties and arbitrary weights, they give a resource augmentation result which achieves a competitive ratio of $O((\log W + \log C)^2)$ when using a slightly faster processor (precisely, $(1+\epsilon)$-speed processor for any $\epsilon > 0$), where W is the max-min ratio of job weights and C is the max-min ratio of job penalties. They also show a lower bound of $\Omega(\max(n^{\frac{1}{4}}, C^{\frac{1}{2}}))$ without using a faster processor, where n is the number of jobs in the job sequence. Note that for the special case when each job has infinite penalty, no jobs would be rejected and the problem reduces to the classic problem of minimizing weighted flow time only. In this case, Becchetti et al. [8] showed a better resource augmentation result, achieving $O(1)$-competitiveness for weighted flow time when using $(1+\epsilon)$-speed processor.

In this paper, we extend the results on rejection penalty [4] in two different directions. First of all, we improve the upper bound result on arbitrary penalties and arbitrary weights. Our online algorithm is constant competitive when using a $(1+\epsilon)$-speed processor, where the constant does not depend on W and C. In other words, for the special case when jobs must be all completed (with infinite penalty), our new algorithm has a comparable performance (but with a larger constant) as Becchetti et al's algorithm [8].

All the above results assume the processor running at a fixed speed. The main results in this paper are on extending the above study of rejection penalty to the dynamic speed scaling model [15] and taking energy into consideration. Specifically, it is assumed that the processor can vary its speed dynamically between 0 and some maximum speed T, and the power P increases with the speed s according to a certain function, say, $P(s) = s^3$. In this setting, a scheduling algorithm has to control job admission and determine the order and speed of job execution, and we are interested to measure the user cost as well as the total energy usage. Note that minimizing user cost and minimizing energy are orthogonal objectives. In this paper, we consider the problem of minimizing a linear combination of the user cost and energy, or simply the user cost plus energy. This problem can also be considered as a generalization of the existing work on minimizing weighted flow time plus energy where job rejection is not allowed [2, 9, 6, 13, 7, 3] (see related work below).

Speed scaling results. For jobs with uniform penalty and unit weight, we give a 6-competitive algorithm for minimizing the user cost plus energy. This algorithm ensures that the penalty of rejected jobs is always at most the flow time plus energy incurred thus far. Intuitively, it maintains a good balance between the flow time plus energy and the penalty. Next, we consider jobs with arbitrary penalties. We show a lower bound result illustrating the problem of minimizing the user cost plus energy being fundamentally more difficult than that of flow time plus energy. Specifically, we assume that $P(s) = s^\alpha$ for some $\alpha > 1$ and jobs have unit weight, and we show that any online algorithm has a competitive ratio of $\Omega(\alpha^{1/2-\epsilon})$ where ϵ is arbitrarily small, even if the maximum speed T is unbounded. This lower bound implies that the competitive ratio must grow with the steepness of the power function (α), while the problem of minimizing flow time plus energy admits a 2-competitive algorithm for any arbitrary power function [7, 3]. We turn to resource augmentation and consider giving the online algorithm a more energy-efficient processor which, using the power $P(s)$, can run at speed $(1+\epsilon)s$ for some $\epsilon > 0$. We call such a processor a $(1+\epsilon)$-speedup processor, based on which we devise an online algorithm for the arbitrary-penalty and arbitrary-weight setting. For any power function $P(s)$, our new algorithm is $O(1)$-competitive for minimizing the user cost plus energy when using a $(1+\epsilon)$-speedup processor. This algorithm, unlike the first one, rejects jobs only at their arrival time and therefore never wastes energy on rejected jobs.

Amortization and potential functions have become standard tool for analyzing algorithms for minimizing flow time plus energy (e.g., [9, 13, 7, 3]). When jobs can be rejected, the online algorithm A and the optimal offline algorithm OPT may have completed two different sets of jobs. This complicates the analysis. For the case of uniform penalty and unit weight, the potential analysis only allows us to bound the flow time plus energy incurred by some special active jobs in A in terms of the cost of OPT. For cost incurred by other active jobs, our technique is an accounting argument to upper bound this cost of A by the total penalty of OPT. For arbitrary penalty and arbitrary weight jobs, taking the advantage of the use of speedup processor, we can directly incorporate the job penalty into the potential analysis and the maximum speed constraint T into the potential function.

Related work on dynamic speed scaling. To reduce energy usage, major chip manufacturers like Intel and IBM are now producing processors that can support *dynamic speed scaling*, which allows operating systems to manage the power by scaling the processor speed dynamically. How to exploit speed scaling effectively has become an interesting problem for the algorithmic community. Yao et al. [15] were the first to consider online job scheduling that takes speed scaling and energy usage into consideration. They considered a model where a processor can vary its speed s, and the energy is consumed at the rate s^α for some constant $\alpha > 1$ (in CMOS based processors, α is believed to be 3 [5]). Running jobs slower saves energy, yet it takes longer time. The challenge arises from the conflicting objectives of optimizing energy usage and some quality of service such as flow time. To understand their tradeoff, Albers and Fujiwara [2] initiated the study of minimizing a linear combination of the total flow and energy. The intuition is that, from an economic viewpoint, users are willing to pay a certain (say, ρ) units of energy to reduce one unit of flow time. By changing the units of time and energy, one can further assume $\rho = 1$ and thus wants to minimize flow plus energy. Following Albers and Fujiwara's work, there is a chain of work on speed scaling algorithms [2, 9, 6, 13, 7, 3], gradually improving the competitive ratios as well as dropping the assumptions on the speed-to-power functions. Now the best known algorithms can work for any arbitrary power function. For jobs with unit weight, a 2-competitive algorithm has been obtained [3]. For arbitrary weight, a competitive ratio of $O(1 + \frac{1}{\epsilon})$ can be achieved using a $(1 + \epsilon)$-speedup processor [7, 11].

Power functions and notations. Throughout the paper, we assume $P(0) = 0$, and P is defined, strictly increasing, strictly convex, continuous and differentiable at all speeds in $[0, T]$; if $T = \infty$, the speed range is $[0, \infty)$ and for any speed x, there exists x' such that $P(x)/x < P(s)/s$ for all $s > x'$ (otherwise the optimal speed scaling policy is to always run at the infinite speed and an optimal schedule is not well-defined). We use Q to denote P^{-1}. Note that Q is strictly increasing and concave. E.g., if $P(s) = s^\alpha$, then $Q(x) = x^{1/\alpha}$. For each job j, we use $p(j)$, $w(j)$ and $v(j)$ to denote its work, weight and penalty.

Organization of the paper. The following discussion focuses on the results on the dynamic speed scaling model only. Our improved result on minimizing the user cost alone on a fixed-speed processor would be shown as a special case in Section 3. Section 2 considers jobs with uniform penalty and unit weight and presents a 6-competitive algorithm for minimizing the user cost plus energy. Section 3 gives the results on jobs with arbitrary penalties and arbitrary weights. Finally, a lower bound result is given in Section 4.

2 Uniform Penalty and Unit Weight

This section considers jobs with the same penalty $c > 0$ and unit weight. We give an online algorithm UPUW for minimizing flow plus penalty plus energy. The job rejection policy of UPUW is similar to that in [4], but the involvement of speed scaling and energy complicates the analysis and demands a potential function. Our main result is the following theorem.

▶ **Theorem 1.** *Consider jobs with uniform penalty and unit weight. Algorithm* UPUW *is 6-competitive for minimizing flow plus penalty plus energy.*

Algorithm UPUW. At time t, let $n_{\mathrm{a}}(t)$ and $s_{\mathrm{a}}(t)$ be respectively the number of active jobs (i.e., jobs that have been released but not yet finished) and the speed of UPUW. Recall that Q is the inverse of the power function P. We set $s_{\mathrm{a}}(t) = \min(Q(n_{\mathrm{a}}(t)+1), T)$. UPUW always runs the job with the smallest remaining work (SRPT) at speed $s_{\mathrm{a}}(t)$ (ties are broken by job ids). Let ϕ be a counter that counts the flow plus energy incurred until time t, i.e., $\phi(t) = \int_0^t (n_{\mathrm{a}}(x) + P(s_{\mathrm{a}}(x)))\,\mathrm{d}x$. Whenever ϕ crosses a multiple of c, UPUW rejects the active job with the largest remaining work (if $n_{\mathrm{a}}(t) > 0$).

To prove Theorem 1, we compare UPUW with the optimal offline schedule OPT. Consider any job sequence. Let G_{a} be the total flow plus energy of UPUW and R_{a} be the total penalty of UPUW, and similarly define G_{o} and R_{o} for OPT. Let t_e be the time when all jobs are completed or rejected by both UPUW and OPT. By definition of UPUW, $G_{\mathrm{a}} = \phi(t_e)$ and $R_{\mathrm{a}} \le G_{\mathrm{a}}$. Thus, we have

▶ **Fact 2.** *The flow plus penalty plus energy of* UPUW *is* $G_{\mathrm{a}} + R_{\mathrm{a}} \le 2\phi(t_e)$.

To upper bound $\phi(t_e)$, we define another counter ψ such that at any time t, $\psi(t) \ge \phi(t)$. Then it suffices to upper bound $\psi(t_e)$ by the cost of OPT. We define ψ as follows: Initially, $\psi(0) = 0$. Whenever OPT rejects a job at t, ψ increases by c. At other times, if $\psi = \phi$, ψ increases at the same rate as ϕ, else (i.e., $\psi > \phi$), ψ stays the same.

Analysis framework. We will upper bound $\psi(t_e)$ in terms of G_{o} and R_{o}. Note that $\psi(t)$ is non-decreasing and increases in two cases: (Case 1) ψ increases by c whenever OPT rejects a job, and (Case 2) ψ increases at the same rate as ϕ whenever $\psi(t) = \phi(t)$. The increase due to Case 1 is bounded by R_{o}. To bound the increase due to Case 2, at any time t, we define a special subset of active jobs, denoted $B(t)$, as follows. Let $k(t) = \lfloor \frac{\psi(t)}{c} \rfloor - \lfloor \frac{\phi(t)}{c} \rfloor$. Let $B(t)$ be the set of the $n_{\mathrm{a}}(t) - k(t)$ active jobs in UPUW with the smallest remaining work ($B(t) = \emptyset$ if $n_{\mathrm{a}}(t) < k(t)$), and let $\tilde{n}_{\mathrm{a}}(t)$ denote the size of $B(t)$. Whenever $\psi(t) = \phi(t)$, $k(t) = 0$. If $\tilde{n}_{\mathrm{a}}(t) = 0$, it implies $n_{\mathrm{a}}(t) \le 0$, and $\phi(t)$ as well as $\psi(t)$ do not increase. Thus, the increase of ψ due to Case 2 is bounded by the flow plus energy incurred by UPUW during times when $\tilde{n}_{\mathrm{a}}(t) \ge 1$, which is upper bounded in Lemma 3. To prove Lemma 3, we follow the analysis in [7, 3] but adapt the potential function to focus only on jobs in $B(t)$ instead of all active jobs in UPUW. Like [7, 3], the potential analysis requires an upper bound on $\tilde{n}_{\mathrm{a}}(t) - n_{\mathrm{o}}(t)$ at any time t, where $n_{\mathrm{o}}(t)$ is the number of active jobs in OPT (Lemma 5). Yet with job rejections, we need new technique to obtain such upper bound, which will be proven via a mapping of jobs in $B(t)$ to jobs in a schedule related to OPT (Lemma 6).

We first state Lemma 3 and show how this lemma leads to Theorem 1. For any time interval I, let $G_{\mathrm{a}}[I]$ and $G_{\mathrm{o}}[I]$ be the flow plus energy incurred during I by UPUW and OPT, respectively.

▶ **Lemma 3.** *For any time interval* $I = (t_1, t_2)$ *such that* $\tilde{n}_{\mathrm{a}}(t_1) = \tilde{n}_{\mathrm{a}}(t_2) = 0$ *and* $\tilde{n}_{\mathrm{a}}(t) > 0$ *for any* $t \in I$, $G_{\mathrm{a}}[I] \le 3 \cdot G_{\mathrm{o}}[I] + 2\int_{t\in I} k(t)\,\mathrm{d}t$.

Proof of Theorem 1. Let $I_1, I_2, ..., I_m$ be all the intervals in $[0, t_e]$ such that for each $I_i = (t_1, t_2)$, $\tilde{n}_{\mathrm{a}}(t_1) = \tilde{n}_{\mathrm{a}}(t_2) = 0$ and $\tilde{n}_{\mathrm{a}}(t) > 0$ for any $t \in I_i$. Let $S = I_1 \cup I_2 \cup \cdots \cup I_m$. Recall that the increase due to Case 2 (in the analysis framework above) can only happen when $\tilde{n}_{\mathrm{a}} \ge 1$, i.e. only during S. Then, by Lemma 3, the increase of ψ due to Case 2 is at most $\int_{t\in S} n_{\mathrm{a}}(t) + P(s_{\mathrm{a}}(t))\,\mathrm{d}t = \sum_{i=1}^m G_{\mathrm{a}}[I_i] \le \sum_{i=1}^m \left(3 \cdot G_{\mathrm{o}}[I_i] + 2\int_{t\in I_i} k(t)\,\mathrm{d}t\right) \le 3 \cdot G_{\mathrm{o}} + 2\int_{t\in S} k(t)\,\mathrm{d}t$.

We now upper bound $\int_{t\in S} k(t)\,\mathrm{d}t$. Whenever OPT rejects a job, ψ increases by c, and then ψ stays the same until ϕ reaches ψ. Since $\phi(t)$ increases at the rate of $n_{\mathrm{a}}(t)+P(s_{\mathrm{a}}(t))$, we have $R_{\mathrm{o}} = \int_{t:\psi(t)>\phi(t)} n_{\mathrm{a}}(t) + P(s_{\mathrm{a}}(t))\,\mathrm{d}t$. Note that $k(t) = \lfloor \frac{\psi(t)}{c} \rfloor - \lfloor \frac{\phi(t)}{c} \rfloor > 0$ implies $\psi(t) > \phi(t)$. Thus, $R_{\mathrm{o}} \geq \int_{t:k(t)>0} n_{\mathrm{a}}(t) + P(s_{\mathrm{a}}(t))\,\mathrm{d}t$. At any time $t \in S$, $\tilde{n}_{\mathrm{a}}(t) > 0$ and hence $k(t) < n_{\mathrm{a}}(t)$. Then $\int_{t\in S} k(t)\,\mathrm{d}t = \int_{t:t\in S\wedge k(t)>0} k(t)\,\mathrm{d}t < \int_{t:t\in S\wedge k(t)>0} n_{\mathrm{a}}(t)\,\mathrm{d}t \leq \int_{t:k(t)>0} n_{\mathrm{a}}(t)\,\mathrm{d}t < R_{\mathrm{o}}$.

Therefore, the increase of ψ due to Case 2 is at most $3G_{\mathrm{o}} + 2R_{\mathrm{o}}$. Adding up the increase of ψ due to Case 1, i.e., R_{o}, gives $\psi(t_e) \leq 3G_{\mathrm{o}} + 3R_{\mathrm{o}}$. By Fact 2 and $\phi(t_e) \leq \psi(t_e)$, $G_{\mathrm{a}} + R_{\mathrm{a}} \leq 6(G_{\mathrm{o}} + R_{\mathrm{o}})$ and hence UPUW is 6-competitive. ◀

The rest of the section is devoted to proving Lemma 3. Before giving the potential analysis, we state a property of set $B(t)$ and show that at any time t, the size of $B(t)$ is no more than the number of active jobs in OPT by $P(T) - 1$ (Lemma 5). Without loss of generality, we assume $P(T) \geq 1$,[1] and OPT rejects a job only at its arrival time.

▶ **Property 4.** *At any time t, the set $B(t)$ only changes upon various events as follows.*

(i) *If a job j arrives and* OPT *rejects j, then $n_{\mathrm{a}} - k$ remains the same, so either B does not change or j replaces another job j' with remaining work at least $p(j)$ in B.*

(ii) *If a job j arrives and* OPT *admits j, $n_{\mathrm{a}} - k$ increases by 1, so either B remains empty, or j is added to B, or another job j' with remaining work at most $p(j)$ is added to B.*

(iii) *If* UPUW *completes a job j, $n_{\mathrm{a}} - k$ decreases by 1, so either B remains empty or j leaves B.*

(iv) *If* UPUW *rejects a job, then ϕ reaches a multiple of c, and $n_{\mathrm{a}} - k$ either remains the same or decreases by 1. Thus, either B does not change or a job in B leaves B.*

▶ **Lemma 5.** *At any time t, $\tilde{n}_{\mathrm{a}}(t) - n_{\mathrm{o}}(t) + 1 \leq P(T)$.*

We now show Lemma 5. If $\tilde{n}_{\mathrm{a}}(t) = 0$, $\tilde{n}_{\mathrm{a}}(t) = 0 \leq P(T) - 1$ (as $P(T) \geq 1$). If $s_{\mathrm{a}}(t) < T$, then $s_{\mathrm{a}}(t) = Q(n_{\mathrm{a}}(t) + 1) < T$ and hence $n_{\mathrm{a}}(t) + 1 \leq P(T)$, so $\tilde{n}_{\mathrm{a}}(t) \leq n_{\mathrm{a}}(t) \leq P(T) - 1$.

It remains to consider that $\tilde{n}_{\mathrm{a}}(t) \geq 1$ and $s_{\mathrm{a}}(t) = T$. Let t' be the last time before t such that $\tilde{n}_{\mathrm{a}}(t') = 0$ or $s_{\mathrm{a}}(t') < T$. By above, we can show that $\tilde{n}_{\mathrm{a}}(t') \leq P(T) - 1$. For any time $x \in (t', t]$, $\tilde{n}_{\mathrm{a}}(x) \geq 1$ and $s_{\mathrm{a}}(x) = T$. Let $N_o(x)$ be the set of jobs arriving during $(t', x]$ that are admitted by OPT. Suppose OPT has completed h jobs in $N_o(t)$ in $(t', t]$. Let S be the schedule obtained by running SRPT at speed T during $(t', t]$ on jobs $B(t') \cup N_o(t)$. Since SRPT maximizes the number of jobs completed by any time [14], S completes at least h jobs during $(t', t]$. As $n_{\mathrm{o}}(t) \geq |N_o(t)| - h$, the number of active jobs in S at t is at most $\tilde{n}_{\mathrm{a}}(t') + |N_o(t)| - h \leq P(T) - 1 + n_{\mathrm{o}}(t)$.

We relate the schedule of UPUW with S. At any time $x \in [t', t]$, let $B(x) = \{j_1, j_2, \cdots, j_{\tilde{n}_{\mathrm{a}}(x)}\}$, ordered in non-decreasing remaining work in UPUW; we always use job ids for tie-breaking. We can show Lemma 6 below by induction on time $[t', t]$ over various events stated in Property 4. Details will be given in the full paper. This lemma implies that at time t, the size of $B(t)$ is less than the number of active jobs in S, i.e., $\tilde{n}_{\mathrm{a}}(t) \leq P(T) - 1 + n_{\mathrm{o}}(t)$, implying Lemma 5.

▶ **Lemma 6.** *At any time $x \in (t', t]$, there is a one-to-one mapping $\rho : B(x) \to B(t') \cup N_o(x)$ such that the remaining work of each $j_i \in B(x)$ in* UPUW *is at most that of $\rho(j_i)$ in S, and $\rho(j_1), \rho(j_2), \cdots, \rho(j_{\tilde{n}_{\mathrm{a}}(x)})$ are in non-decreasing order of remaining work in S.*

[1] If $P(T) < 1$, we use the algorithm in [4] for job selection, which is 2-competitive for flow plus penalty, and always run at speed T. When the algorithm is running a job, the power is less than 1 and the number of active jobs is at least 1, so the total energy usage is at most the total flow time and hence this algorithm is 4-competitive.

We now give the potential analysis for proving Lemma 3. Recall that we are considering an interval $I = (t_1, t_2)$. Let $G_{\mathrm{a}}(t)$ and $G_{\mathrm{o}}(t)$ be the flow plus energy incurred from time t_1 up to time t by UPUW and OPT, respectively, for any $t \in I$. It suffices to define a potential function $\Phi(t)$ for any time $t \in I$ such that the following conditions hold: *(i) Boundary condition:* At time t_1 and t_2, $\Phi = 0$. *(ii) Discrete-event condition:* During I, when a job arrives, or a job is completed by UPUW or OPT, or a job is rejected by UPUW, $\Delta\Phi(t) \le 0$. *(iii) Running condition:* At any other time $t \in I$, $\frac{\mathrm{d}G_{\mathrm{a}}(t)}{\mathrm{d}t} + \frac{\mathrm{d}\Phi(t)}{\mathrm{d}t} \le 3 \cdot \frac{\mathrm{d}G_{\mathrm{o}}(t)}{\mathrm{d}t} + 2k(t)$. Then, Lemma 3 follows by integrating these conditions over I.

Potential function $\Phi(t)$. Consider any time t. Let $\tilde{n}_{\mathrm{a}}(q,t)$ and $n_{\mathrm{o}}(q,t)$ be the number of active jobs in $B(t)$ and OPT, respectively, with remaining work at least q. Note that $\tilde{n}_{\mathrm{a}}(t) = \tilde{n}_{\mathrm{a}}(0,t)$ and $n_{\mathrm{o}}(t) = n_{\mathrm{o}}(0,t)$. We will drop the parameter t from the notations when t refers to the current time clearly. Let $(\cdot)_+ = \max(\cdot, 0)$. We adapt the potential function given in [7, 3] as follows:

$$\Phi(t) = 3\int_0^\infty \sum_{i=1}^{(\tilde{n}_{\mathrm{a}}(q,t)-n_{\mathrm{o}}(q,t))_+} P'(Q(i))\,\mathrm{d}q\ .$$

The boundary condition holds because at t_1 and t_2, $\tilde{n}_{\mathrm{a}} = 0$, so $\tilde{n}_{\mathrm{a}}(q) = 0$ for all q and $\Phi = 0$. We now check the discrete-event condition. Note that $\tilde{n}_{\mathrm{a}}(x) \ge 1$ for any $x \in (t_1, t_2)$. When a job j arrives and is rejected by OPT, by Property 4(i), there are two cases: (Case 1) B does not change, then Φ does not change. (Case 2) j replaces another job j' with remaining work $q' \ge p(j)$ in B. Then $n_{\mathrm{a}}(q)$ decreases by 1 for $q \in [p(j), q']$ and Φ does not increase. When a job j arrives and is admitted by OPT, by Property 4(ii), a job j' (which may be j) with remaining work $q' \le p(j)$ is added to B. Then $\tilde{n}_{\mathrm{a}}(q)$ increases by 1 for $q \in [0, q'] \subseteq [0, p(j)]$ and $n_{\mathrm{o}}(q)$ increases by 1 for $q \in [0, p(j)]$. Thus, $\tilde{n}_{\mathrm{a}}(q) - n_{\mathrm{o}}(q)$ does not increase for all q and Φ does not increase. When a job is completed by UPUW or OPT, $\tilde{n}_{\mathrm{a}}(q)$ or $n_{\mathrm{o}}(q)$ changes only at the single point $q = 0$, which does not affect the integration and hence Φ remains the same. Finally, when a job is rejected by UPUW, either B does not change or a job in B leaves B. For the former case, Φ does not change. For the latter case, let q' be the remaining work of the job that leaves B. Then $\tilde{n}_{\mathrm{a}}(q)$ decreases by 1 for $q \in [0, q']$, and hence Φ does not increase.

It remains to check the running condition. Consider any time $t \in (t_1, t_2)$ without job arrival, completion and rejection. Let s_{a} and s_{o} be the current speeds of UPUW and OPT, respectively. To bound the rate of change of Φ, Lemma 7 below shows how Φ changes in an infinitesimal amount of time (from t to $t + \mathrm{d}t$). Its proof is based on similar arguments as in [7, 3] and will be given in the full paper.

► **Lemma 7.** *Consider any time t without job arrival or completion and $\tilde{n}_{\mathrm{a}} \ge 1$. If $\tilde{n}_{\mathrm{a}} < n_{\mathrm{o}}$, then $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \le 0$; if $\tilde{n}_{\mathrm{a}} \ge n_{\mathrm{o}}$, then either* (i) $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \le 3 \cdot P'(Q(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}}))(-s_{\mathrm{a}} + s_{\mathrm{o}})$, *or* (ii) $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \le 3 \cdot P'(Q(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}} + 1))(-s_{\mathrm{a}} + s_{\mathrm{o}})$ *and $n_{\mathrm{o}} \ge 1$, or* (iii) $\frac{\mathrm{d}\Phi}{\mathrm{d}t} = 0$ *and $\tilde{n}_{\mathrm{a}} = n_{\mathrm{o}}$.*

► **Lemma 8.** *At any time in (t_1, t_2) without job arrival, completion and rejection, $\frac{\mathrm{d}G_{\mathrm{a}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \le 3 \cdot \frac{\mathrm{d}G_{\mathrm{o}}}{\mathrm{d}t} + 2k$.*

Proof. Note that during (t_1, t_2), $\tilde{n}_{\mathrm{a}} \ge 1$ and $\tilde{n}_{\mathrm{a}} = n_{\mathrm{a}} - k$. Also, $s_{\mathrm{a}} = \min(Q(n_{\mathrm{a}} + 1), T)$ and hence $\frac{\mathrm{d}G_{\mathrm{a}}}{\mathrm{d}t} = n_{\mathrm{a}} + P(s_{\mathrm{a}}) \le 2n_{\mathrm{a}} + 1 \le 2n_{\mathrm{a}} + \tilde{n}_{\mathrm{a}} = 3\tilde{n}_{\mathrm{a}} + 2k$. Similarly, $\frac{\mathrm{d}G_{\mathrm{o}}}{\mathrm{d}t} = n_{\mathrm{o}} + P(s_{\mathrm{o}})$. If $\tilde{n}_{\mathrm{a}} < n_{\mathrm{o}}$, by Lemma 7, $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \le 0$ and thus $\frac{\mathrm{d}G_{\mathrm{a}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \le 3\tilde{n}_{\mathrm{a}} + 2k < 3n_{\mathrm{o}} + 2k \le 3\frac{\mathrm{d}G_{\mathrm{o}}}{\mathrm{d}t} + 2k$. Otherwise, if $\tilde{n}_{\mathrm{a}} \ge n_{\mathrm{o}}$, we consider the three cases in Lemma 7, where we need the upper bound on $\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}}$ (Lemma 5).

Case (i): $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \le 3P'(Q(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}}))(-s_{\mathrm{a}} + s_{\mathrm{o}})$. By a lemma given in [7] (stated as Lemma 9 below), $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \le 3(-s_{\mathrm{a}} + Q(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}}))P'(Q(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}})) + 3P(s_{\mathrm{o}}) - 3(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}})$. If $s_{\mathrm{a}} = T$, by Lemma 5,

$s_{\mathrm{a}} = T \geq Q(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}} + 1) \geq Q(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}})$; otherwise, $s_{\mathrm{a}} = Q(n_{\mathrm{a}} + 1) \geq Q(\tilde{n}_{\mathrm{a}} + 1) \geq Q(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}})$. Thus, $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 3(n_{\mathrm{o}} + P(s_{\mathrm{o}})) - 3\tilde{n}_{\mathrm{a}}$ and hence $\frac{\mathrm{d}G_{\mathrm{a}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 3\tilde{n}_{\mathrm{a}} + 2k + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 3(n_{\mathrm{o}} + P(s_{\mathrm{o}})) + 2k = 3\frac{\mathrm{d}G_{\mathrm{o}}}{\mathrm{d}t} + 2k$.

Case (ii): $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 3P'(Q(n_{\mathrm{a}} - n_{\mathrm{o}} + 1))(-s_{\mathrm{a}} + s_{\mathrm{o}})$ and $n_{\mathrm{o}} \geq 1$. By Lemma 9, $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 3(-s_{\mathrm{a}} + Q(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}} + 1))P'(Q(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}} + 1)) + 3P(s_{\mathrm{o}}) - 3(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}} + 1)$. If $s_{\mathrm{a}} = T$, by Lemma 5, $s_{\mathrm{a}} = T \geq Q(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}} + 1)$; otherwise, $s_{\mathrm{a}} = Q(n_{\mathrm{a}} + 1) \geq Q(\tilde{n}_{\mathrm{a}} + 1) \geq Q(\tilde{n}_{\mathrm{a}} - n_{\mathrm{o}} + 1)$. Thus, $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 3(n_{\mathrm{o}} + P(s_{\mathrm{o}})) - 3\tilde{n}_{\mathrm{a}} - 3$, and hence $\frac{\mathrm{d}G_{\mathrm{a}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 3\tilde{n}_{\mathrm{a}} + 2k + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 3(n_{\mathrm{o}} + P(s_{\mathrm{o}})) + 2k - 3 \leq 3\frac{\mathrm{d}G_{\mathrm{o}}}{\mathrm{d}t} + 2k$.

Case (iii): $\frac{\mathrm{d}\Phi}{\mathrm{d}t} = 0$ and $\tilde{n}_{\mathrm{a}} = n_{\mathrm{o}}$. Then $\frac{\mathrm{d}G_{\mathrm{a}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 3\tilde{n}_{\mathrm{a}} + 2k = 3n_{\mathrm{o}} + 2k \leq 3\frac{\mathrm{d}G_{\mathrm{o}}}{\mathrm{d}t} + 2k$. ◀

Below is the lemma given in [7], which is used in the proof of Lemma 8.

▶ **Lemma 9.** [7] *Let P be a strictly increasing, strictly convex, continuous and differentiable function. Let $i, s_{\mathrm{a}}, s_{\mathrm{o}} \geq 0$ be any real. Then, $P'(Q(i))(-s_{\mathrm{a}} + s_{\mathrm{o}}) \leq (-s_{\mathrm{a}} + Q(i))P'(Q(i)) + P(s_{\mathrm{o}}) - i$.*

3 Arbitrary Penalty and Arbitrary Weight

This section considers jobs of arbitrary penalty and arbitrary weight in the following two models. In the *fixed-speed* model, the processor always runs at speed 1 and energy is not a concern. The objective is to minimize the user cost, i.e., total weighted flow plus penalty. In the *speed scaling* model, the processor can scale its speed with an arbitrary power function $P(s)$ and maximum speed T. Then the objective is to minimize the user cost plus energy.

In the speed scaling model, we give an $O((1+\frac{1}{\epsilon})^2)$-competitive algorithm for weighted flow plus penalty plus energy, using $(1+\epsilon)^2$-speedup processor for any $\epsilon > 0$. In the fixed-speed model, we give a $(1+\epsilon)^2$-speed $O((1+\frac{1}{\epsilon})^2)$-competitive algorithm for weighted flow plus penalty. This improves the $(1+\epsilon)$-speed $O(\frac{1}{\epsilon}(\log W + \log C)^2)$-competitive result in [4].

Fractional weighted flow. To obtain these results, we will first focus on the objective of total *fractional* weighted flow, and then convert the result for (integral) weighted flow. At any time t, the fractional weight of an active job j, denoted by $w(j,t)$, is its weight times its remaining fraction, i.e., $w(j,t) = w(j) \cdot \frac{q(j,t)}{p(j)}$, where $q(j,t)$ is the remaining size of j at t. Then the fractional weighted flow of job j is $\int_{r(j)}^{\infty} w(j,t)\,\mathrm{d}t$, and hence the total fractional weighted flow is $\int_0^{\infty} w_{\mathrm{a}}(t)\,\mathrm{d}t$, where $w_{\mathrm{a}}(t)$ is the total fractional weight of active jobs at time t.

HDF and future cost. Under a fixed speed function, HDF (highest density first) minimizes fractional weighted flow [8]. Our algorithm will always rejects a job at its arrival time and processes the admitted jobs using HDF. Furthermore, at any time, the processor always scales its speed according to the total fractional weight w of the active jobs, and we denote this speed by $s(w)$ (for fixed-speed processor, $s(w)$ is a constant). Consider any time t. Let $w_{\mathrm{a}}(q,t)$ be the total fractional weight of active jobs with inverse density at least q. Then we can define a *future cost* $\widehat{\Phi}_a(t)$ to capture the total fractional weighted flow to serve the current active jobs if no jobs arrive in the future [12]:

$$\widehat{\Phi}_a(t) = \int_{q=0}^{\infty} \int_{x=0}^{w_{\mathrm{a}}(q,t)} \frac{x}{s(x)} \,\mathrm{d}x\,\mathrm{d}q\ .$$

Algorithm HDF-AC. We focus on the objective of fractional weighted flow and define the algorithm HDF-AC that works for both the speed scaling and fixed-speed models. Let $\epsilon > 0$ be a constant. Consider any time t.

Job execution: Let $w_{\mathrm{a}}(t)$ and $s_{\mathrm{a}}(t)$ be the total fractional weight of active jobs and the speed of HDF-AC. In the fixed-speed model, we use $(1+\epsilon)$-speed processor, so $s_{\mathrm{a}}(t) = 1+\epsilon$;

in the speed scaling model, we use $(1+\epsilon)$-speedup processor and set $s_{\mathrm{a}}(t) = (1+\epsilon)\cdot\min(Q(w_{\mathrm{a}}(t)),T)$. Then, HDF-AC runs the admitted jobs using HDF at speed $s_{\mathrm{a}}(t)$.

Admission control: Let $w_{\mathrm{a}}(q,t)$ be the total fractional weight of active jobs with inverse density at least q. Let $f(x) = \frac{x}{\min(Q(x),T)}$ in the speed scaling model, and $f(x) = x$ in the fixed-speed model. Then the future cost at time t is

$$\widehat{\Phi}_a(t) = \frac{1}{1+\epsilon}\cdot\int_{q=0}^{\infty}\int_{x=0}^{w_{\mathrm{a}}(q,t)} f(x)\,\mathrm{d}x\,\mathrm{d}q\ .$$

When a job j arrives, let $\Delta\widehat{\Phi}_a(t)$ be the increase in $\widehat{\Phi}_a(t)$ if j is admitted. More precisely, let $d(j) = p(j)/w(j)$ be the inverse density of j. Then $\Delta\widehat{\Phi}_a(t) = \frac{1}{1+\epsilon}\cdot\int_{q=0}^{d(j)}\int_{x=w_{\mathrm{a}}(q,t)}^{w_{\mathrm{a}}(q,t)+w(j)} f(x)\,\mathrm{d}x\,\mathrm{d}q$. HDF-AC discards j if $v(j) \le \Delta\widehat{\Phi}_a(t)$; otherwise, j is admitted.

Our main result is the following theorem.

▶ **Theorem 10.** *Consider any $\epsilon > 0$.* (i) *In the speed scaling model, HDF-AC is $(8+\frac{12}{\epsilon})$-competitive for fractional weighted flow plus penalty plus energy, when using $(1+\epsilon)$-speedup processor.* (ii) *In the fixed-speed model, HDF-AC is $(3+\frac{6}{\epsilon})$-competitive for fractional weighted flow plus penalty, when using $(1+\epsilon)$-speed processor.*

Though the objectives for Theorem 10 (i) and (ii) are different, we present an analysis framework that works for both objectives. Let OPT be the optimal offline schedule for the corresponding objective. Without loss of generality, we can assume that OPT rejects a job at its arrival. Let $w_{\mathrm{o}}(t)$ and $s_{\mathrm{o}}(t)$ be the total fractional weight of active jobs and the speed of OPT. In the speed scaling model, the objective is fractional weighted flow plus penalty plus energy. We further assume that OPT runs the BCP algorithm [7], i.e., at any time t, OPT runs the admitted jobs using HDF at speed $s_{\mathrm{o}}(t) = \min(Q(w_{\mathrm{o}}(t)),T)$. Since BCP is 2-competitive for fractional weighted flow plus energy [7], such assumption on OPT only increases the competitive ratio by a factor of 2. In the fixed-speed model, the objective is fractional weighted flow plus penalty. We further assume that OPT runs HDF at speed $s_{\mathrm{o}}(t) = 1$, since HDF minimizes fractional weighted flow [8].

Since OPT runs HDF, we can define its future cost similarly. At any time t, let $w_{\mathrm{o}}(q,t)$ be the total fractional weight of active jobs with inverse density at least q. Recall that $f(x) = \frac{x}{\min(Q(x),T)}$ in the speed scaling model, and $f(x) = x$ in the fixed-speed model. Then the future cost of OPT at time t is

$$\widehat{\Phi}_o(t) = \int_{q=0}^{\infty}\int_{x=0}^{w_{\mathrm{o}}(q,t)} f(x)\,\mathrm{d}x\,\mathrm{d}q\ .$$

Overview of analysis. Our analysis exploits amortization and potential functions. We split the objective into two parts; R denotes the penalty and G denotes the fractional weighted flow (plus energy). Let $G_{\mathrm{a}}(t)$ and $G_{\mathrm{o}}(t)$ denote the objective G incurred up to time t by HDF-AC and OPT, respectively. Define $R_{\mathrm{a}}(t)$ and $R_{\mathrm{o}}(t)$ similarly for the penalty R. To show that HDF-AC is (c_1+c_2)-competitive for the objective $G+R$ against OPT, it suffices to define a potential function $\Phi(t)$ such that the following conditions hold: *(i) Boundary condition:* $\Phi = 0$ before any job is released and after all jobs are completed. *(ii) Completion condition:* When a job is completed by HDF-AC or OPT, $\Delta\Phi(t) \le 0$. *(iii) Arrival condition:* When a job arrives, $\Delta R_{\mathrm{a}}(t) + \Delta\Phi(t) \le c_1\cdot(\Delta\widehat{\Phi}_o(t) + \Delta R_{\mathrm{o}}(t))$, where $\Delta\widehat{\Phi}_o(t)$ is the change in the future cost of OPT at time t. *(iv) Running condition:* At any other time, $\frac{\mathrm{d}G_{\mathrm{a}}(t)}{\mathrm{d}t} + \frac{\mathrm{d}\Phi(t)}{\mathrm{d}t} \le c_2\cdot\frac{\mathrm{d}G_{\mathrm{o}}(t)}{\mathrm{d}t}$.

To see the correctness, note that $R_{\mathrm{a}}(t)$ and $R_{\mathrm{o}}(t)$ changes discretely only at job arrivals, and $G_{\mathrm{a}}(t)$ and $G_{\mathrm{o}}(t)$ changes continuously at other times. Let t_e be the time when all jobs are completed by both HDF-AC and OPT. Since the future cost $\widehat{\Phi}_o(t)$ captures the fractional weighted flow incurred by OPT to serve the active jobs at t, we have $\int_0^{t_e} \Delta\widehat{\Phi}_o(t)\,\mathrm{d}t \leq G_\mathrm{o}(t_e)$. Therefore, the correctness follows from integrating these conditions over time, which gives

$$G_\mathrm{a}(t_e) + R_\mathrm{a}(t_e) \leq c_1 \cdot (\int_0^{t_e} \Delta\widehat{\Phi}_o(t)\,\mathrm{d}t + R_\mathrm{o}(t_e)) + c_2 \cdot G_\mathrm{o}(t_e) \leq (c_1 + c_2) \cdot (G_\mathrm{o}(t_e) + R_\mathrm{o}(t_e)) \ .$$

Potential function. We now define a general form of $\Phi(t)$ that works for both objectives. Consider any time t. Recall that $f(x) = \frac{x}{\min(Q(x),T)}$ in the speed scaling model, and $f(x) = x$ in the fixed-speed model. The potential function Φ is defined as

$$\Phi(t) = \frac{2}{\epsilon} \cdot \int_{q=0}^{\infty} \int_{x=0}^{(w_\mathrm{a}(q,t)-w_\mathrm{o}(q,t))_+} f(x)\,\mathrm{d}x\,\mathrm{d}q \ .$$

The boundary and completion conditions hold obviously. We now check the arrival condition. We drop the parameter t from all notations when it is clear that t refers to the current time.

▶ **Lemma 11.** *When a job j arrives, $\Delta R_\mathrm{a} + \Delta\Phi \leq (2 + \frac{4}{\epsilon}) \cdot (\Delta\widehat{\Phi}_o + \Delta R_\mathrm{o})$.*

Proof. Let $d(j) = p(j)/w(j)$ be the inverse density of job j. If HDF-AC admits this job, then $w_\mathrm{a}(q)$ increases by $w(j)$ for $q \in [0, d(j)]$. Similarly, if OPT admits this job, then $w_\mathrm{o}(q)$ increases by $w(j)$ for $q \in [0, d(j)]$. Now, we consider the following two cases.

Case 1: OPT admits j. In this case, $\Delta R_\mathrm{o} = 0$. If HDF-AC also admits j, then $w_\mathrm{a}(q) - w_\mathrm{o}(q)$ remains the same for all q, so $\Delta\Phi = 0$. Since $\Delta\widehat{\Phi}_o \geq 0$, $\Delta R_\mathrm{a} + \Delta\Phi = 0 \leq (2 + \frac{4}{\epsilon}) \cdot (\Delta\widehat{\Phi}_o + \Delta R_\mathrm{o})$.

Otherwise, HDF-AC rejects j. We analyze using techniques in [12]. Note that $\Delta\widehat{\Phi}_o = \int_{q=0}^{d(j)} \int_{x=w_\mathrm{o}(q)}^{w_\mathrm{o}(q)+w(j)} f(x)\,\mathrm{d}x\,\mathrm{d}q$. The change of Φ due to OPT is

$$-\frac{2}{\epsilon} \cdot \int_{q=0}^{d(j)} \int_{x=(w_\mathrm{a}(q)-w_\mathrm{o}(q)-w(j))_+}^{(w_\mathrm{a}(q)-w_\mathrm{o}(q))_+} f(x)\,\mathrm{d}x\,\mathrm{d}q \ .$$

Note that $\Delta R_\mathrm{a} = v(j) \leq \frac{1}{1+\epsilon} \int_{q=0}^{d(j)} \int_{x=w_\mathrm{a}(q)}^{w_\mathrm{a}(q)+w(j)} f(x)\,\mathrm{d}x\,\mathrm{d}q \leq \frac{2}{\epsilon} \int_{q=0}^{d(j)} \int_{x=w_\mathrm{a}(q)}^{w_\mathrm{a}(q)+w(j)} f(x)\,\mathrm{d}x\,\mathrm{d}q$, and the change of Φ due to HDF-AC is zero. Thus,

$$\Delta R_\mathrm{a} + \Delta\Phi \leq \frac{2}{\epsilon} \cdot \int_{q=0}^{d(j)} \left(\int_{x=w_\mathrm{a}(q)}^{w_\mathrm{a}(q)+w(j)} f(x)\,\mathrm{d}x - \int_{x=(w_\mathrm{a}(q)-w_\mathrm{o}(q)-w(j))_+}^{(w_\mathrm{a}(q)-w_\mathrm{o}(q))_+} f(x)\,\mathrm{d}x \right) \mathrm{d}q \ .$$

It was shown in [12] that if the function f satisfies that $f(0) \geq 0$ and f is increasing and subadditive, i.e., for any $a, b \geq 0$, $f(a+b) \leq f(a) + f(b)$, then $\int_{x=w_\mathrm{a}(q)}^{w_\mathrm{a}(q)+w(j)} f(x)\,\mathrm{d}x - \int_{x=(w_\mathrm{a}(q)-w_\mathrm{o}(q)-w(j))_+}^{(w_\mathrm{a}(q)-w_\mathrm{o}(q))_+} f(x)\,\mathrm{d}x \leq 2 \int_{x=w_\mathrm{o}(q)}^{w_\mathrm{o}(q)+w(j)} f(x)\,\mathrm{d}x$. In the fixed-speed model, $f(x) = x$ obviously satisfies these conditions. In the speed scaling model, $f(x) = \frac{x}{\min(Q(x),T)}$. It was also shown in [12] that $\frac{x}{Q(x)}$ is increasing and subadditive. Clearly, $f(x)$ is also increasing. Consider any $a, b \geq 0$. If $a + b \leq P(T)$, it follows directly that $f(a+b) \leq f(a) + f(b)$; otherwise, $f(a+b) = \frac{a+b}{T} \leq \frac{a}{\min(Q(a),T)} + \frac{b}{\min(Q(b),T)} = f(a) + f(b)$. Therefore, in both cases, we can apply the inequality to get that $\Delta R_\mathrm{a} + \Delta\Phi \leq \frac{4}{\epsilon} \cdot \int_{q=0}^{d(j)} \int_{x=w_\mathrm{o}(q)}^{w_\mathrm{o}(q)+w(j)} f(x)\,\mathrm{d}x\,\mathrm{d}q = \frac{4}{\epsilon} \cdot \Delta\widehat{\Phi}_o \leq (2 + \frac{4}{\epsilon}) \cdot (\Delta\widehat{\Phi}_o + \Delta R_\mathrm{o})$.

Case 2: OPT rejects j. In this case, $\Delta R_\mathrm{o} = v(j)$, $\Delta\widehat{\Phi}_o = 0$, and the change of Φ due to OPT is zero. Similarly, if HDF-AC admits j, then $\Delta R_\mathrm{a} = 0$ and the change of Φ due to HDF-AC is

$$\frac{2}{\epsilon} \cdot \int_{q=0}^{d(j)} \int_{x=w_\mathrm{a}(q)}^{w_\mathrm{a}(q)+w(j)} f(x)\,\mathrm{d}x\,\mathrm{d}q \ ,$$

which is exactly $2(1 + \frac{1}{\epsilon})$ times the increase of $\widehat{\Phi}_a$ and is therefore at most $2(1 + \frac{1}{\epsilon})v(j)$. Otherwise, if HDF-AC also rejects j, $\Delta R_\mathrm{a} = v(j)$ and the change of Φ due to HDF-AC is zero. In both cases, $\Delta R_\mathrm{a} + \Delta\Phi \leq (2 + \frac{2}{\epsilon}) \cdot v(j) \leq (2 + \frac{4}{\epsilon}) \cdot (\Delta\widehat{\Phi}_o + \Delta R_\mathrm{o})$. ◀

It remains to show the running condition. Consider any time t without job arrival or completion. Let s_{a} and s_{o} be the current speeds of HDF-AC and OPT, respectively. To bound the rate of change of Φ, Lemma 12 below shows how Φ changes in an infinitesimal amount of time (from t to $t+\mathrm{d}t$). Its proof is based on similar arguments as in [7, 12] and will be given in the full paper.

▶ **Lemma 12.** *Consider any time without job arrival or completion.* (i) *If $w_{\mathrm{a}} < w_{\mathrm{o}}$, then $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 0$.* (ii) *If $w_{\mathrm{a}} > w_{\mathrm{o}}$, then $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq \frac{2}{\epsilon} \cdot f(w_{\mathrm{a}} - w_{\mathrm{o}}) \cdot (-s_{\mathrm{a}} + s_{\mathrm{o}})$.* (iii) *If $w_{\mathrm{a}} = w_{\mathrm{o}}$, then $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq \frac{2}{\epsilon} \cdot f(w_{\mathrm{o}}) \cdot s_{\mathrm{o}}$.*

We are ready to show the running condition for the speed scaling model (Lemma 13) and for the fixed-speed model (Lemma 14).

▶ **Lemma 13.** *In the speed scaling model, at any time without job arrival or completion, $\frac{\mathrm{d}G_{\mathrm{a}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq (2 + \frac{2}{\epsilon}) \cdot \frac{\mathrm{d}G_{\mathrm{o}}}{\mathrm{d}t}$.*

Proof. Since $s_{\mathrm{a}} = (1+\epsilon)\min(Q(w_{\mathrm{a}}), T) \leq (1+\epsilon)Q(w_{\mathrm{a}})$ and HDF-AC is using a $(1+\epsilon)$-speedup processor, $P(s_{\mathrm{a}}) \leq w_{\mathrm{a}}$ and $\frac{\mathrm{d}G_{\mathrm{a}}}{\mathrm{d}t} \leq 2w_{\mathrm{a}}$. By the assumption of OPT, $s_{\mathrm{o}} = \min(Q(w_{\mathrm{o}}), T)$ and $\frac{\mathrm{d}G_{\mathrm{o}}}{\mathrm{d}t} \geq w_{\mathrm{o}}$. We now consider the three cases stated in Lemma 12. Recall that $f(x) = \frac{x}{\min(Q(x),T)}$.
Case (i): $w_{\mathrm{a}} < w_{\mathrm{o}}$. By Lemma 12, $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 0$, so $\frac{\mathrm{d}G_{\mathrm{a}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 2w_{\mathrm{a}} < 2w_{\mathrm{o}} \leq (2+\frac{2}{\epsilon}) \cdot \frac{\mathrm{d}G_{\mathrm{o}}}{\mathrm{d}t}$.
Case (ii): $w_{\mathrm{a}} > w_{\mathrm{o}}$. Note that Q is increasing. By Lemma 12, $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq \frac{2}{\epsilon} \cdot \frac{w_{\mathrm{a}} - w_{\mathrm{o}}}{\min(Q(w_{\mathrm{a}} - w_{\mathrm{o}}),T)}(-(1+\epsilon)\min(Q(w_{\mathrm{a}}),T) + \min(Q(w_{\mathrm{o}}),T)) \leq -\frac{2}{\epsilon} \cdot (w_{\mathrm{a}} - w_{\mathrm{o}}) \frac{\epsilon \cdot \min(Q(w_{\mathrm{a}}),T)}{\min(Q(w_{\mathrm{a}} - w_{\mathrm{o}}),T)} \leq 2w_{\mathrm{o}} - 2w_{\mathrm{a}}$. Thus, $\frac{\mathrm{d}G_{\mathrm{a}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 2w_{\mathrm{a}} + 2w_{\mathrm{o}} - 2w_{\mathrm{a}} \leq (2+\frac{2}{\epsilon}) \cdot \frac{\mathrm{d}G_{\mathrm{o}}}{\mathrm{d}t}$.
Case (iii): $w_{\mathrm{a}} = w_{\mathrm{o}}$. By Lemma 12, $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq \frac{2}{\epsilon} \cdot \frac{w_{\mathrm{o}}}{\min(Q(w_{\mathrm{o}}),T)} \cdot \min(Q(w_{\mathrm{o}}),T) = \frac{2}{\epsilon} \cdot w_{\mathrm{o}}$. Thus, $\frac{\mathrm{d}G_{\mathrm{a}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 2w_{\mathrm{a}} + \frac{2}{\epsilon} \cdot w_{\mathrm{o}} = (2+\frac{2}{\epsilon}) \cdot w_{\mathrm{o}} \leq (2+\frac{2}{\epsilon}) \cdot \frac{\mathrm{d}G_{\mathrm{o}}}{\mathrm{d}t}$. ◀

▶ **Lemma 14.** *In the fixed-speed model, at any time without job arrival or completion, $\frac{\mathrm{d}G_{\mathrm{a}}}{\mathrm{d}t} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq (1 + \frac{2}{\epsilon}) \cdot \frac{\mathrm{d}G_{\mathrm{o}}}{\mathrm{d}t}$.*

Proof. It suffices to show that $w_{\mathrm{a}} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq (1+\frac{2}{\epsilon}) \cdot w_{\mathrm{o}}$. Recall that $s_{\mathrm{a}} = 1+\epsilon$, $s_{\mathrm{o}} = 1$ and $f(x) = x$. We now consider the three cases stated in Lemma 12.
Case (i): $w_{\mathrm{a}} < w_{\mathrm{o}}$. By Lemma 12, $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq 0$, so $w_{\mathrm{a}} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq w_{\mathrm{a}} \leq w_{\mathrm{o}} \leq (1+\frac{2}{\epsilon}) \cdot w_{\mathrm{o}}$.
Case (ii): $w_{\mathrm{a}} > w_{\mathrm{o}}$. By Lemma 12, $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq \frac{2}{\epsilon} \cdot (w_{\mathrm{a}} - w_{\mathrm{o}}) \cdot (-(1+\epsilon)+1) = 2w_{\mathrm{o}} - 2w_{\mathrm{a}}$. Therefore, $w_{\mathrm{a}} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq w_{\mathrm{a}} + 2w_{\mathrm{o}} - 2w_{\mathrm{a}} \leq w_{\mathrm{o}} \leq (1+\frac{2}{\epsilon}) \cdot w_{\mathrm{o}}$.
Case (iii): $w_{\mathrm{a}} = w_{\mathrm{o}}$. By Lemma 12, $\frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq \frac{2}{\epsilon} \cdot w_{\mathrm{o}}$. Thus, $w_{\mathrm{a}} + \frac{\mathrm{d}\Phi}{\mathrm{d}t} \leq w_{\mathrm{a}} + \frac{2}{\epsilon} \cdot w_{\mathrm{o}} = (1+\frac{2}{\epsilon}) \cdot w_{\mathrm{o}}$. ◀

In the speed scaling model, by Lemmas 11 and 13, HDF-AC is $(4+\frac{6}{\epsilon})$-competitive against OPT. Recall that OPT uses BCP and thus is 2-approximate. Therefore, Theorem 10 (i) follows. In the fixed-speed model, by Lemmas 11 and 14, HDF-AC is $(3+\frac{6}{\epsilon})$-competitive against OPT, which is the actual optimal schedule. Thus, Theorem 10 (ii) follows.

Online algorithm for integral weighted flow. We now convert the result of Theorem 10 for the objective of (integral) weighted flow. Since HDF is $(1+\epsilon)$-speed $(1+\frac{1}{\epsilon})$-competitive for weighted flow on a fixed speed processor [8] and the fractional weighted flow of any schedule (including OPT) is always at most its (integral) weighted flow, we use the following online algorithm HDF-AC*: HDF-AC* keeps a simulated copy of HDF-AC on the same job instance. It always follows the admission control of HDF-AC. At any time, HDF-AC* runs at speed $(1+\epsilon)$ faster than the simulated HDF-AC, but selects the job to run using HDF on its own active jobs.

The following performance guarantee of HDF-AC* follows directly from Theorem 10.

▶ **Corollary 15.** *Consider any $\epsilon > 0$.* (i) *In the speed scaling model, HDF-AC* is* $(1 + \frac{1}{\epsilon})(8 + \frac{12}{\epsilon})$*-competitive for weighted flow plus penalty plus energy, when using* $(1+\epsilon)^2$*-speedup processor.* (ii) *In the fixed-speed model, HDF-AC* is* $(1 + \frac{1}{\epsilon})(3 + \frac{6}{\epsilon})$*-competitive for weighted flow plus penalty, when using* $(1+\epsilon)^2$*-speed processor.*

4 Lower Bound for Arbitrary Penalty Jobs

This section gives the lower bound result. Assuming $P(s) = s^\alpha$, we show that the competitive ratio of any algorithm must grow with α, i.e., the steepness of the power function. This implies that no $O(1)$-competitive algorithm exists for arbitrary power function.

▶ **Theorem 16.** *Consider minimizing flow plus energy plus penalty. For power function* $P(s) = s^\alpha$*, if* T *is unbounded, any algorithm is* $\Omega(\alpha^{1/2-\epsilon})$*-competitive for any* $0 < \epsilon < \frac{1}{2}$*.*

Proof. Let A be any algorithm and OFF be the offline adversary. Let $k \geq 1$ be some constant depending on α (to be defined later). At time 0, the adversary releases two streams of jobs, namely Stream 1 and Stream 2. Stream 1 contains k^2 jobs of size 1 and penalty k^2, each released at time i, where $0 \leq i \leq k^2 - 1$. Stream 2 contains k job of size k and penalty k^5, each released at time jk, where $0 \leq j \leq k - 1$. The penalty of Stream 2 jobs is large enough such that A is not competitive if any one of them is rejected. Therefore, A runs Stream 2 jobs one by one (in SRPT) in their arrival order. Depending on the number of Stream 2 jobs remaining in A at time k^2, the adversary may release Stream 3, which contains $\frac{k^4}{\delta}$ job of size $\delta = \frac{1}{k}$ and penalty k^5, each released at time $k^2 + i\delta$, where $0 \leq i \leq \frac{k^4}{\delta} - 1$.

Case 1: At time k^2, A has less than $\frac{k}{2}$ Stream 2 jobs remaining. In this case, the adversary does not release Stream 3. OFF can always run at speed 1 and completes the Stream 1 jobs one by one in $[0, k^2]$ and then completes the Stream 2 jobs one by one in $[k^2, 2k^2]$. Thus, the total flow of OFF is at most $k^2 \cdot 1 + k \cdot (k^2 + k) = O(k^3)$. Since OFF always consume power $1^\alpha = 1$, which is at most the number of active jobs at that time, the energy usage of OFF is at most its flow. As OFF does not reject any job, the flow plus energy plus penalty of OFF is $O(k^3)$.

Consider the schedule of A. If A rejects at least one Stream 2 jobs, the penalty of A is at least k^5. If A rejects more than $\frac{k^2}{4}$ Stream 1 jobs, the penalty of A is at least $\frac{k^2}{4} \cdot k^2 = \Omega(k^4)$. If A has at least $\frac{k^2}{8}$ Stream 1 jobs remaining at time k^2, the flow of these jobs is at least $\sum_{i=1}^{k^2/8} i = \Omega(k^4)$. In all of the above three cases, the competitive ratio of A is $\Omega(k)$. Otherwise, A does not reject any Stream 2 job, and A rejects at most $\frac{k^2}{4}$ Stream 1 jobs, and there are less than $\frac{k^2}{8}$ Stream 1 jobs remaining at time k^2. Thus, during $[0, k^2]$, A has completed at least $k^2 - \frac{k^2}{4} - \frac{k^2}{8} = \frac{5k^2}{8}$ Stream 1 jobs and at least $k - \frac{k}{2} = \frac{k}{2}$ Stream 2 jobs. The work done of A during $[0, k^2]$ is at least $\frac{5k^2}{8} + \frac{k}{2} \cdot k = \frac{9k^2}{8}$. By the convexity of the power function s^α, running at a fixed speed minimizes the energy usage and thus the energy usage of A is at least $(\frac{9k^2}{8}/k^2)^\alpha \cdot k^2 = (\frac{9}{8})^\alpha k^2$. Thus, the competitive ratio of A is $\Omega((\frac{9}{8})^\alpha \cdot \frac{1}{k})$.

Case 2: At time k^2, A has at least $\frac{k}{2}$ Stream 2 jobs remaining. In this case, the adversary releases Stream 3. Similar to Stream 2, without loss of generality, A works on Stream 3 jobs one by one (in SRPT). OFF can reject all Stream 1 jobs and then always run at speed 1 to complete the Stream 2 jobs one by one in $[0, k^2]$ and then completes the Stream 3 jobs one by one in $[k^2, k^2 + k^4]$. Thus, the total penalty of OFF is $k^2 \cdot k^2 = k^4$ and total flow of OFF is at most $k \cdot k + \frac{k^4}{\delta} \cdot \delta = k^2 + k^4 = O(k^4)$. Since OFF always consume power $1^\alpha = 1$, which is at most the number of active jobs at that time, the energy usage of OFF is at most its flow. Therefore, the flow plus energy plus penalty of OFF is $O(k^4)$.

Consider the schedule of A. If A rejects at least one Stream 2 or Stream 3 job, the penalty of A is at least k^5. If at time k^2+k^4, A has at least $\frac{k}{4}$ Stream 2 jobs remaining, the flow of these jobs is at least $\frac{k}{4}\cdot k^4=\Omega(k^5)$. If at time k^2+k^4, A has at least $\frac{k^2}{8\delta}$ Stream 3 jobs remaining, the flow of these jobs is at least $\delta\cdot\sum_{i=1}^{k^2/8\delta} i=\Omega(\frac{k^4}{\delta})=\Omega(k^5)$. In all of the above three cases, the competitive ratio of A is $\Omega(k)$. Otherwise, A does not reject any Stream 2 and Stream 3 job, and at time k^2+k^4, there are less than $\frac{k}{4}$ Stream 2 jobs and less than $\frac{k^2}{8\delta}$ Stream 3 jobs remaining. Thus, A has completed more than $\frac{k}{2}-\frac{k}{4}=\frac{k}{4}$ Stream 2 jobs and more than $\frac{k^4}{\delta}-\frac{k^2}{8\delta}$ Stream 3 jobs during $[k^2,k^2+k^4]$. Since A runs Stream 2 jobs and Stream 3 jobs by SRPT, respectively, the total work done during $[k^2,k^2+k^4]$ is at least $\frac{k}{4}\cdot k+(\frac{k^4}{\delta}-\frac{k^2}{8\delta})\cdot\delta=k^4+\frac{k^2}{8}$. Since running at a fixed speed minimizes the energy usage, the energy usage of A is at least $k^4\cdot((k^4+\frac{k^2}{8})/k^4)^\alpha=\Omega(k^4\cdot(1+\frac{1}{8k^2})^\alpha)$ and hence A is $\Omega((1+\frac{1}{8k^2})^\alpha)$-competitive.

Therefore, A is $\Omega(\min(k,(\frac{9}{8})^\alpha(\frac{1}{k}),(1+\frac{1}{8k^2})^\alpha))$-competitive. We set $k=\alpha^{\frac{1}{2}-\epsilon}$ for $0<\epsilon<\frac{1}{2}$. Since $(1+\frac{1}{8y})^y$ is increasing with y, the competitive ratio of A is $\Omega(\min(\alpha^{\frac{1}{2}-\epsilon},(\frac{9}{8})^\alpha/\alpha^{\frac{1}{2}-\epsilon},(1+\frac{1}{8\alpha^{1-2\epsilon}})^{\alpha^{1-2\epsilon}\cdot\alpha^{2\epsilon}}))=\Omega(\min(\alpha^{\frac{1}{2}-\epsilon},(\frac{9}{8})^\alpha/\alpha^{\frac{1}{2}-\epsilon},(1+\frac{1}{8})^{\alpha^{2\epsilon}}))=\Omega(\alpha^{\frac{1}{2}-\epsilon})$. ◀

References

1 S. Albers. Energy-efficient algorithms. *Communications of the ACM*, 53(5):86–96, 2010.

2 S. Albers and H. Fujiwara. Energy-efficient algorithms for flow time minimization. *ACM Transactions on Algorithms*, 3(4):49, 2007.

3 L. Andrew, A. Wierman, and A. Tang. Optimal speed scaling under arbitrary power functions. *ACM SIGMETRICS Performance Evaluation Review*, 37(2):39–41, 2009.

4 N. Bansal, A. Blum, S. Chawla, and K. Dhamdhere. Scheduling for flow-time with admission control. In *Proc. ESA*, pages 43–54, 2003.

5 D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J. D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook. Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. *IEEE Micro*, 20(6):26–44, 2000.

6 N. Bansal, H. L. Chan, T. W. Lam, and L. K. Lee. Scheduling for speed bounded processors. In *Proc. ICALP*, pages 409–420, 2008.

7 N. Bansal, H. L. Chan, and K. Pruhs. Speed scaling with an arbitrary power function. In *Proc. SODA*, pages 693–701, 2009.

8 L. Becchetti, S. Leonardi, A. Marchetti-Spaccamela, and K. Pruhs. Online weighted flow time and deadline scheduling. *J. Discrete Algorithms*, 4(3):339–352, 2006.

9 N. Bansal, K. Pruhs, and C. Stein. Speed scaling for weighted flow time. *SIAM Journal on Computing*, 39(4):1294–1308, 2009.

10 H. L. Chan, J. Edmonds, T. W. Lam, L. K. Lee, A. Marchetti-Spaccamela, and K. Pruhs. Nonclairvoyant speed scaling for flow and energy. In *Proc. STACS*, pages 255–264, 2009.

11 S. H. Chan, T. W. Lam, and L. K. Lee. Non-clairvoyant speed scaling for weighted flow time. In *Proc. ESA*, pages 23–35, 2010.

12 A. Gupta, R. Krishnaswamy, and K. Pruhs. Scalably scheduling power-heterogeneous processors. In *Proc. ICALP*, 312–323, 2010.

13 T. W. Lam, L. K. Lee, I. To, and P. Wong. Speed scaling functions for flow time scheduling based on active job count. In *Proc. ESA*, pages 647–659, 2008.

14 L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

15 F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *Proc. FOCS*, pages 374–382, 1995.

Clique-width: When Hard Does Not Mean Impossible*

Robert Ganian[1], Petr Hliněný[1], and Jan Obdržálek[1]

1 **Masaryk University, Brno**
{ganian,hlineny,obdrzalek}@fi.muni.cz

Abstract

In recent years, the parameterized complexity approach has lead to the introduction of many new algorithms and frameworks on graphs and digraphs of bounded clique-width and, equivalently, rank-width. However, despite intensive work on the subject, there still exist well-established hard problems where neither a parameterized algorithm nor a theoretical obstacle to its existence are known. Our article is interested mainly in the digraph case, targeting the well-known Minimum Leaf Out-Branching (cf. also Minimum Leaf Spanning Tree) and Edge Disjoint Paths problems on digraphs of bounded clique-width with non-standard new approaches.

The first part of the article deals with the Minimum Leaf Out-Branching problem and introduces a novel XP-time algorithm wrt. clique-width. We remark that this problem is known to be W[2]-hard, and that our algorithm does not resemble any of the previously published attempts solving special cases of it such as the Hamiltonian Path. The second part then looks at the Edge Disjoint Paths problem (both on graphs and digraphs) from a different perspective – rather surprisingly showing that this problem has a definition in the MSO_1 logic of graphs. The linear-time FPT algorithm wrt. clique-width then follows as a direct consequence.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases clique-width, bi-rank-width, minimum leaf out-branching, minimum leaf spanning tree, edge-disjoint paths

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.404

1 Introduction

It is known that the majority of graph problems are NP-complete in general, so alternative approaches are necessary for tackling these problems. The utilization of parameterized algorithmics is one such very successful approach, where instead of focusing on the general class of all graphs we design algorithms on graphs with a bounded structural parameter (or "width"). This has strong practical motivation, since real-world applications generally work with specific classes of graphs as input.

"Polynomial runtime" parameterized algorithms are roughly divided into two groups. The more ideal case constitutes *fixed-parameter tractable* (FPT) algorithms, where the runtime is $poly(n) \cdot f(k)$ (n being the input size and k the parameter). Unfortunately, not all combinations of problems and parameters allow FPT algorithms, and so in some cases it is necessary to settle for an *XP algorithm* – i.e. an algorithm with runtime $poly(n)^{f(k)}$. Notice that the exponent in XP algorithms increases with the parameter, but still the runtime remains polynomial for any fixed value of k.

* This work was supported by the Czech research grants GAČR 201/09/J021 and 202/11/0196.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 404–415
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

As for the parameters themselves, the one best known today is the tree-width of Robertson and Seymour [17] which has allowed for efficient solution of many NP-hard problems on all graphs having bounded tree-width. The drawback is that the class of graphs with bounded tree-width is quite restrictive. A lot of research since then has focused on obtaining a width measure which would be more general and still allow efficient algorithms for a wide range of NP-hard problems on graphs of bounded width. This has lead to the introduction of clique-width by Courcelle and Olariu [4] and, subsequently, of rank-width by Oum and Seymour [16]. Both of these width parameters are related in the sense that one is bounded if and only if the other is bounded. We refer to Section 2 for further details.

In this article, we provide polynomial algorithms for two well-established problems on digraphs of bounded clique-width/bi-rank-width.

- The first one is *Minimum Leaf Out-Branching*, a problem which generalizes the Hamiltonian Path problem and which is studied e.g. in [5]. The task is to find a spanning out-tree in a digraph that minimizes the number of leaves. Definition and more details are provided in Section 3. We remark that the undirected variant is known as Minimum Leaf Spanning Tree problem (e.g. [19]), and our results apply also to that.
- The second one is *Edge Disjoint Paths* problem, asking for pairwise edge-disjoint paths between a fixed number of terminal pairs. In this case the directed variant is much more difficult than the undirected one – see details in Section 4.

Parameterized complexity status of Minimum Leaf Out-Branching remained unsolved in our previous work on digraphs of bounded bi-rank-width [9], resisting the dynamic programming approaches traditionally used e.g. for clique-width. The provided new Algorithm 12 in Section 3 solves the problem and is also straightforwardly applicable to undirected graphs.

▶ **Theorem 1** (Algorithm 12). *The Minimum Leaf Out-Branching problem on a given digraph G of clique-width k (with arbitrary number of leaves) can be solved in XP time $\mathcal{O}(n^{f(k)})$, where $f(k) \sim 2^{\mathcal{O}(k)}$ if a k-expression of G is given, and $f(k) \sim 2^{\mathcal{O}(2^k)}$ otherwise.*

The second part of the article shortly deals with the Edge Disjoint Paths problem with a fixed number of paths. Note that this was the only remaining open (directed) variant of Disjoint Paths with respect to parameterization by clique-width – see [9, 12, 15] for complexity results and/or algorithms for the other variants. We show in Section 4 that even the Edge Disjoint Paths problem may be described by an MSO_1 formula. This is somehow surprising given the fact that MSO_1 cannot speak about sets of edges, and our logical formula is definitely not a trivial restatement of the original problem. In the end we obtain, in connection with [3]:

▶ **Theorem 2** (Theorem 17). *Both the undirected and directed variant of the Edge Disjoint Paths problem with a fixed number of terminal pairs have a linear-time FPT algorithm on simple (di)graphs of bounded clique-width.*

Theorem 2 can, moreover, be directly used as a subroutine in a new algorithm for the Edge Disjoint Paths problem on tournaments by Chudnovsky and Seymour [in preparation].

2 Clique-width and rank-width

We use standard graph and digraph (directed graph) notation. All our graphs and digraphs are simple (i.e. do not contain loops or multiple edges) unless specified otherwise.

▶ **Definition 3** (clique-width, [4]). Let k be a positive integer. A pair (G, γ) is a *k-labelled graph* if G is a graph and $\gamma : V(G) \to \{1, 2, \ldots, k\}$ is a mapping. We call $\gamma(v)$ for $v \in V(G)$ the *label* of a vertex v. As γ is usually fixed, we often write just G for the k-labelled graph (G, γ), and we refer to $\gamma(v)$ as to the G-label of a vertex v. A *k-expression* is a well formed expression t built using the four operators defined below. Let $1 \leq i, j \leq k$. Then

1. $[i]$ is a nullary operator which represents a graph with a single vertex labelled i,
2. $\eta_{i,j}$, for $i \neq j$, is a unary operator which adds edges between all pairs of vertices where one is labelled i and the other is labelled j,
3. $\rho_{i\to j}$ is a unary operator which changes the labels of all vertices labelled i to j, and
4. $\oplus$ is a binary operator which represents disjoint union of two k-labelled graphs.

Each k-expression t naturally corresponds to (*generates*) a k-labelled graph G which will be denoted for reference by $G[t] = G$. The *clique-width* of an undirected graph G is then the smallest k such that there exists a k-expression generating G. For digraphs clique-width is defined in just the same way, only the operator $\eta_{i,j}$ is replaced by the operator $\alpha_{i,j}$ which creates directed edges (arcs) from each vertex with label i to each vertex with label j. It is known [4] that every graph of clique-width k can be generated by an *irredundand* expression, i.e. an expression that applies the $\eta_{i,j}$ / $\alpha_{i,j}$ operator only in situations when there is no edge from a vertex of label i to one of label j.

It is quite natural to view a k-expression t_G corresponding to G as a tree T with nodes labelled by subterms of t_G (t_G being the root), together with a bijection between the leaves of the tree and vertices of G. In this setting the *type* of each node $t \in V(T)$ is the top-level operator of t, and so we have four different node types.

▶ **Example 4.** $\alpha_{1,2}(\rho_{1\to2}(\alpha_{1,2}(\rho_{1\to2}(\alpha_{1,2}([1]\oplus[2]))\oplus[1]))\oplus[1])$ is a 2-expression corresponding to a directed clique of size 4. See Fig. 1.

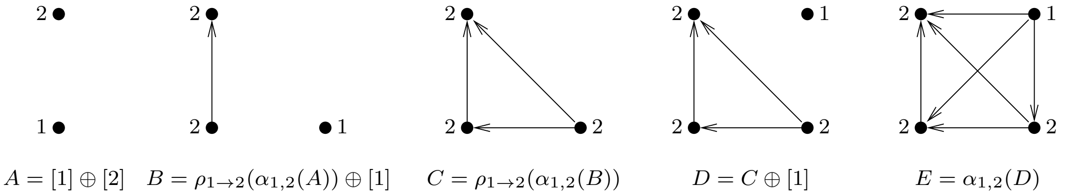

2
1
2
2
1
2
2
2
2
1
2
2
2
1
2
2
$A = [1] \oplus [2]$
$B = \rho_{1\to2}(\alpha_{1,2}(A)) \oplus [1]$
$C = \rho_{1\to2}(\alpha_{1,2}(B))$
$D = C \oplus [1]$
$E = \alpha_{1,2}(D)$

Figure 1 Construction of the directed clique of size 4

Closely related to clique-width is another structural parameter, called *rank-width* [16] (on undirected graphs) or *bi-rank-width* [14] (on digraphs). Due to space restrictions we only refer to [11] for their definitions. The relationship of these measures to the former is that they are bounded if and only if clique-width is bounded. However, a crucial advantage of rank-width is that it can be computed optimally by an FPT algorithm. To be more specific:

▶ **Theorem 5** ([2, 16]). *$rwd(G) \leq cwd(G) \leq 2^{rwd(G)+1} - 1$ for all graphs G.*

▶ **Theorem 6** ([13, 14]). *For every integer parameter k there is an $O(n^3)$-time FPT algorithm that, for a given n-vertex graph G, either finds a bi-rank-decomposition of G of width at most k, or confirms that the bi-rank-width of G is more than k.*

Due to lack of space for comprehensible definitions and explanation of rank-width and their parse trees we stick with (perhaps better known) clique-width in this article. All the results, however, could be straightforwardly reformulated for (bi-)rank-width.

3 Minimum leaf out-branching

Let $out_G(x)$ denote the out-degree of x in a digraph G, i.e. the number of edges having their tail in x. For an edge f and nonadjacent vertices x, y of a digraph G, we write $G - f$ to denote the graph resulting by removal f from G, and $G + (x, y)$ for the graph obtained by adding a new edge from x to y. A digraph T is an *out-tree* if T is an oriented tree with only one vertex of in-degree zero (called the *root*). The vertices of out-degree zero are called *leaves* of T. An *out-forest* is a digraph whose weakly connected components are out-trees.

▶ **Definition 7.** Let G be a digraph. We say that T is an *out-branching* of G if T is a spanning subdigraph of G, i.e. $V(T) = V(G)$ and $E(T) \subseteq E(G)$, and T is an out-tree. The *Minimum Leaf Out-Branching problem* (or MinLOB for short) is the problem of deciding, for a digraph G and integer ℓ on the input, whether G contains an out-branching with at most ℓ leaves.

Notice that not every digraph has an out-branching. It is not hard to show that G has an out-branching if, and only if, there is a vertex $v \in V(G)$ such that there is a directed path from v to any vertex of G. This is checkable in linear-time [1], but the MinLOB problem itself is NP-complete since it contains the Hamiltonian Path as a special case ($\ell = 1$). It is also possible to analogically define the *Maximum Leaf Out-Branching problem* (MaxLOB), asking for an out-branching with at least ℓ leaves, but this variant seems to have quite different (and rather easier) algorithmic behaviour than MinLOB.

The core contribution of our paper is to resolve one important question left open in [9]; what is the computational complexity of MinLOB when parameterized by the clique-width / bi-rank-width of the input graph? It follows by a reduction from Hamiltonian Path [7] that MinLOB is W[2]-hard with respect to clique-width (even with fixed ℓ), and so does not have an FPT algorithm unless the Exponential Time Hypothesis fails. The first XP algorithm for the undirected Hamiltonian Path parameterized by the clique-width was due to Espelage et al [6]. Another XP algorithm for ℓ-MinLOB for every fixed ℓ and parameterized by the bi-rank-width has been recently given in [11].

Our new XP algorithm for MinLOB parameterized by the clique-width does not resemble any of the aforementioned algorithms for the Hamiltonian Path and ℓ-MinLOB problems. In fact, our new algorithm seems to be in a certain fundamental aspect (see Theorem 13 and Question 14) very different from the many other parameterized algorithms designed for graphs of bounded clique-width. We suggest that this difference may not be fully understood yet, and so it deserves further conceptual investigation, too.

3.1 Out-branching and modules

We first show some basic properties of the problem as a prelude to the coming algorithm in the next section. Though these properties (cf. Definitions 8, 9) are not directly used in Algorithm 12 and its proof, we consider them worth independent interest.

For a digraph G, a set $M \subseteq V(G)$ is called a *module* if every vertex of M has the same in-neighbourhood and out-neighbourhood (as every other in M) among the vertices not in M. Generalizing the module concept, we consider a k-labelled digraph (H, γ) such that $H \subseteq G$. We say that H is a *labelled-modular subdigraph* of G if $\gamma^{-1}(i)$ is a module in $G - E(H)$ for all $i = 1, 2, \ldots, k$. Note that H is not required to be an induced subdigraph of G.

In other words, H is a labelled-modular subdigraph of G if the existence of an edge in $G - E(H)$ incident with some $v \in V(H)$ "depends only on" the label of v. Notice that if s is a subexpression of a (irredundand) k-expression t, then the generated k-labelled digraph

$G[s]$ is always a labelled-modular subdigraph of the whole $G[t]$ (an analogical claim holds e.g. for bi-rank-decompositions).

Let G be a digraph, $H \subseteq G$ its subgraph and $F \subseteq H$ an out-forest. We call the pair (F, μ) where $\mu : V(H) \to \mathbb{N}$ an *annotated out-forest.* We say that the annotated out-forest (F, μ) *extends* to an out-branching $T \subseteq G$ if $E(F) = E(T) \cap E(H)$, and for all $x \in V(H)$ we have $\mu(x) = out_T(x) - out_F(x)$. We are going to define an equivalence relation $\approx_H$ on the set of all annotated out-forests of a k-labelled graph H, with the intended meaning to "capture all important information" about possible extendability of a particular annotated out-forest into an out-branching.

▶ **Definition 8** (Canonical equivalence)**.** A pair of annotated out-forests (F_1, μ_1) and (F_2, μ_2) in a k-labelled digraph H is canonically equivalent, written as $(F_1, \mu_1) \approx_H (F_2, \mu_2)$, if, and only if, the following holds for each integer ℓ and every digraph G such that H is a labelled-modular subdigraph of G: (F_1, μ_1) can be extended to an out-branching of G with $\leq \ell$ leaves if and only if (F_2, μ_2) can be extended to an out-branching of G with $\leq \ell$ leaves.

On the other hand, in Definition 9 we introduce simple "information about (F, μ)" that is sufficient to determine its equivalence class within $\approx_H$. For every connected component (out-tree) T_0 of F in H, including the isolated vertices of H not incident with any edge of F, the *shape* of T_0 is the pair (a, B) where a is the H-label of the root of T_0 and B is the set of all H-labels occurring at the vertices $x \in V(T_0)$ such that $\mu(x) > 0$ (*active* vertices).

▶ **Definition 9** (Out-forest signatures)**.** The *signature* of a (spanning) annotated out-forest (F, μ) in a k-labelled digraph H is a vector in $\mathbb{N}^*$ consisting of

- the number of leaves x of F (incl. isolated vertices) such that $\mu(x) = 0$,
- for every $i = 1, \dots, k$, the sum of $\mu(x)$ over all vertices $x \in V(H)$ of the H-label i, and
- for every possible shape, the number of out-trees of F having this shape.

Notice that the length of this vector depends only on k and not on the size of H.

▶ **Lemma 10.** *Let (F_1, μ_1) and (F_2, μ_2) be a pair of annotated out-forests in a k-labelled digraph H. If the signatures of (F_1, μ_1) and (F_2, μ_2) are equal, then $(F_1, \mu_1) \approx_H (F_2, \mu_2)$.*

Due to lack of space, we skip the proof of this lemma. The claim clearly suggests that an XP-time algorithm for MinLOB might exist since the information "carried by" the set of available signatures is of polynomial size. Unfortunately, even this strong claim is not strong enough to give such an algorithm (unlike in the finite Myhill–Nerode-type case, e.g. [8], or in many other XP solvable problems [11]) since we do not know how to process available signature vectors dynamically along a k-expression.

3.2 A dynamic algorithm for MinLOB

In order to obtain an XP algorithm for the MinLOB problem, we introduce a "weaker" alternative to Definition 9. Recall that a vertex x of an annotated out-forest (F, μ) is *active* if $\mu(x) > 0$. We now relax this notion to suit the coming algorithm.

Assume a k-expression t generating the digraph $H = G[t]$, an annotated out-forest (F, μ) in H, and a vertex $v_q \in V(H)$ generated by the leaf q of t. We say that v_q is *potentially active* in H for the k-expression t if, for every node (subexpression) s of t on the path from q to the root, the annotated out-forest induced by (F, μ) in $G[s] \subseteq H$ contains an active vertex (possibly v_q itself) of the same $G[s]$-label as that of v_q. In particular, if a vertex x is active in (F, μ), then x is also potentially active for t. If there is no active vertex of label i

in (F, μ), then there is also no such potentially active vertex. It may, however, happen that there are many more potentially active vertices of (F, μ) for t than the active ones.

For every connected component (out-tree) T_0 of F in H, we define the *weak shape* of T_0 as the pair (a, B) where a is the H-label of the root of T_0 and B is the set of all H-labels occurring at the vertices $x \in V(T_0)$ that are potentially active for the k-expression t.

▶ **Definition 11** (Weak signature, cf. Definition 9)**.** The *weak signature* of a spanning annotated out-forest (F, μ) in the k-labelled digraph $H = G[t]$ generated by a k-expression t is a vector $\vec{w}$ in $\mathbb{N}^c$ (with the appropriate length c) consisting of the sections

- wl, the number of leaves x of F (incl. isolated vertices) such that $\mu(x) = 0$,
- $wa(i)$ for every $i = 1, \dots, k$, where $wa(i)$ equals the sum of $\mu(x)$ over all vertices x of the H-label i (informally, the “total multiplicity” of all active vertices of label i), and
- $ws(a, B)$ for every possible weak shape (a, B), equal to the number of out-trees (weak components) of F having this weak shape (a, B) in H for the k-expression t.

The advantage of a weak signature over former signature is that weak signatures are easier to handle in dynamic programming on a k-expression of the input graph. Still, the situation is not as easy as if we could dynamically compute the set of all weak signatures of all possible annotated outforests in our graph — we can only compute a suitable superset of it via the following straightforward algorithm.

▶ **Algorithm 12.** Assume an input consisting of a k-expression t generating a k-labelled digraph G on n vertices. The following algorithm computes, in XP-time wrt. the parameter k, a set $\mathcal{U}$ of vectors from $\mathbb{N}^c$ (cf. Definition 11) such that $\mathcal{U}$ includes all weak signatures of spanning annotated out-forests in G for t.

I. Input t (a k-expression); $G = G[t]$.

II. At every leaf $q = [i]$ of t (where $i \in \{1, \dots, k\}$), the graph $G[q]$ is actually a single vertex v_q of label i. Let U_q be the set of weak signatures of the edge-less annotated out-forests $(G[q], \mu_j)$ where $\mu_j(v_q) = j$, over $0 \le j \le out_G(v_q)$.

III. At every internal node r of t, we compute in the leaves-to-root direction as follows.

$\boldsymbol{r = p \oplus q}$**:** U_r is the set, for all pairs $\vec{c} \in U_p$, $\vec{d} \in U_q$, of their vector sums $\vec{c} + \vec{d}$.

$\boldsymbol{r = \rho_{i \to j}(q)}$**:** We initialize $U_r = \emptyset$. Then, for every $\vec{c} \in U_q$, $\vec{c} = (wl, \vec{wa}, \vec{ws})$ as in Definition 11, we compute; $\vec{wa}' = \vec{wa}$ except that $wa'(j) = wa(j) + wa(i)$ and $wa'(i) = 0$, and $\vec{ws}'$ “shifting” the components of $\vec{ws}$ according to the effect that relabeling $i \to j$ has on all possible weak shapes. We add $(wl, \vec{wa}', \vec{ws}')$ to U_r.

$\boldsymbol{r = \alpha_{i,j}(q)}$**:** We initialize $U_r = U_q$. Then we repeat the following procedure as long as U_r is changing:

- Pick arbitrary $\vec{c} \in U_r$, $\vec{c} = (wl, \vec{wa}, \vec{ws})$ such that $wa(i) > 0$, and any weak shapes (a, B) and (b, C) such that $i \in B$, $b = j$, and $ws(a, B) > 0$, $ws(j, C) > 0$.
- Let $\vec{wa}' = \vec{wa}$ except that $wa'(i) = wa(i) - 1$.
- Let $\vec{ws}' = \vec{ws}$ except that $ws'(a, B) = ws(a, B) - 1$, $ws'(j, C) = ws(j, C) - 1$, and $ws'(a, B \cup C) = ws(a, B \cup C) + 1$.
- If $wa'(i) = 0$, then let the label i be subsequently “removed from” the label sets (of potentially active vertices) of all weak shapes indexing $\vec{ws}'$.
- Finally, add $(wl, \vec{wa}', \vec{ws}')$ to U_r.

IV. Output $\mathcal{U} = U_t$.

Proof. There are two steps in the proof.

Claim. The set $\mathcal{U}$ contains, for every annotated out-forest (F, μ) in G such that $\mu(x) \leq out_G(x) - out_F(x)$, the weak signature of (F, μ) for t.

This is easily proved by leaves-to-root structural induction on t: The claim is trivial at the leaves. Considering a node $r = p \oplus q$, the weak signature of any annotated out-forest in $G[r]$ that is obtained as a disjoint union of annotated out-forests in $G[p], G[q]$ of weak signatures $\vec{c}, \vec{d}$, respectively, equals computed $\vec{c} + \vec{d}$. Analogically for $r = \alpha_{i,j}(q)$. Notice that none of those two operations change potential activity of vertices by definition.

Consider one iteration at a node $r = \alpha_{i,j}(q)$. Let $(F + (u, v), \mu')$ be an annotated out-forest in $G[r]$ such that the weak signature $\vec{c}$ of (F, μ), $\mu(u) = \mu'(u) + 1$, has already been computed in previous iterations of U_r by the inductive assumption. Hence u of $G[r]$-label i is active in (F, μ) and $\vec{c}$ contains a weak shape (a, B) such that $i \in B$. Furthermore, since $F + (u, v)$ is an outforest, $\vec{c}$ contains a weak shape (b, C) such that $b = j$ is the $G[r]$-label of v. Then the vector $(wl, \vec{wa}', \vec{ws}')$ computed by the algorithm from $\vec{c}$ is exactly the weak signature of $(F + (u, v), \mu')$ by definition.

Claim. Algorithm 12 runs in XP time, i.e. in time $\mathcal{O}(n^{f(k)})$ where $f(k) \sim 2^{\mathcal{O}(k)}$.

The runtime of the algorithm is clearly dominated (up to a constant multiple of the exponent) by the number of possible weak signature vectors of length c. It is $c = 1 + k + k2^k$. The value of each vector component may be a natural number up to n for $wl, \vec{ws}$ and up to n^2 for $\vec{wa}$. Hence the claim follows. ◀

The importance of Algorithm 12 comes from the following crucial statement.

▶ **Theorem 13.** *Suppose that the set $\mathcal{U} = U_t$ computed in Algorithm 12 contains a weak signature vector $\vec{w} = (wl, \vec{wa}, \vec{ws})$ such that $wl = \ell$, $\vec{wa} = \vec{0}$, and $\vec{ws}$ containing only one non-zero entry* 1 *(i.e. $\vec{w}$ corresponds to a weak signature of an out-tree with ℓ leaves and zero annotation). Then the graph $G = G[t]$ contains an out-branching with ℓ leaves.*

Consequently, Algorithm 12 solves the Minimum Leaf Out-Branching problem – for a given G and arbitrary ℓ – in XP-time wrt. the clique-width k of G (Theorem 1).

Proof. Let $\vec{c} \in U_s$ be a weak signature vector computed by Algorithm 12 on a subexpression s of the k-expression t. A *derivation tree* δ of $\vec{c}$ over t is a rooted tree which is a subdivision of that of s, and every node of δ is labelled with one weak signature vector as follows:

- Each $r \in V(s) \subseteq V(\delta)$ is labelled with some $\vec{c}_r \in U_r$ such that the root of δ is labelled by $\vec{c}$, and for each edge $(r, q) \in E(s)$ it holds that $\vec{c}_r$ is computed from $\vec{c}_q$ by Algorithm 12.
- Moreover, $(r, q) \in E(\delta)$ unless r is of the form "$r = \alpha_{i,j}(q)$". If $r = \alpha_{i,j}(q)$ and $\vec{c}_r$ was created from $\vec{c}_q$ by adding k edges, then (r, q) is replaced with an r–q-path $(r_0 = r, r_1, \dots, r_k = q)$ of length k in δ such that $\vec{c}_{r_\ell}$, $0 \leq \ell < k$, is obtained from $\vec{c}_{r_{\ell+1}}$ by one (productive) iteration of the "$r = \alpha_{i,j}(q)$" step in III.

Typically, one vector $\vec{c}$ can have many derivation trees.

Such a derivation tree δ is *realizable* over t if there exists an annotated out-forest (F, μ) in $G[s]$ such that, for each node d of δ, the corresponding subforest of (F, μ) has weak signature equal to the label of d. Obviously not all vectors in U_s have realizable derivations, in general.

Let $\mathcal{V} \subseteq \mathcal{U}$ be the set of *good vectors* assumed in the statement of this theorem, i.e. of those vectors $\vec{w} = (wl, \vec{wa}, \vec{ws}) \in \mathcal{U}$ such that $wl = \ell$, $\vec{wa} = \vec{0}$, and $\vec{ws}$ containing only one non-zero entry 1. Among all the good vectors $\vec{w} \in \mathcal{V}$, we select $\vec{w}_0$ and a derivation tree δ_0 of $\vec{w}_0$ such that there is a derivation tree $\delta_1 \subseteq \delta_0$ which is realizable over t and δ_1 maximizes the number of edges of its realizing out-forest (F_1, μ_1). We aim to show, by means of contradiction, that $\delta_1 = \delta_0$. Then (F_1, μ_1) would be a realization of whole δ_0 of weak signature $\vec{w}_0 \in \mathcal{V}$, and hence F_1 is an outbranching with ℓ leaves by the definition of $\mathcal{V}$.

Let $\delta_1 \subsetneq \delta_0$. Analyzing Algorithm 12. III, one easily finds out that both the "$r = p \oplus q$" and "$r = \rho_{i\to j}(q)$" operations preserve realizability. Hence we have got a realizing out-forest (F_1, μ_1) of δ_1 over t, its weak signature $\vec{c}_1$, and the label $\vec{c}_2$ of the parent of the root of δ_1 in the derivation tree δ_0 such that: $\vec{c}_2$ results from $\vec{c}_1$ by one iteration of the "$r = \alpha_{i,j}(q)$" step in III, but no single edge of G can be added to (F_1, μ_1) to produce an out-forest of weak signature $\vec{c}_2$. In the rest of the proof we are going to construct another annotated out-forest with one more edge than (F_1, μ_1) such that its weak signature is contained in the derivation tree of some good vector in $\mathcal{V}$ (and this will be a contradiction to the assumptions).

We need a few more technical terms before proceeding with our proof.

- An *out-branching of a weak signature* vector $\vec{c}$ is any out-tree Γ such that $V(\Gamma)$ is the multiset of weak shapes respecting their multiplicities given by $\vec{c}$, i.e. every weak shape has the appropriate number of unique copies in $V(\Gamma)$. Informally, if $\vec{c}$ were realizable by an out-forest F, then the vertices of Γ would be all the out-trees of F.
- Considering a weak signature $\vec{c}$ labelling a node of a derivation tree δ, we say that an out-branching Γ of $\vec{c}$ is *determined by* δ if the following holds for every pair $x, y \in V(\Gamma)$: $(x, y) \in E(\Gamma)$ iff the computation run of Algorithm 12 associated with δ contains a "directed sequence" of $\alpha_{i,j}$ operations interconnecting the particular copies x to y.
- An out-branching Γ of a weak signature $\vec{c}$ is *feasible* for t if there exists good $\vec{d} \in \mathcal{V}$ such that a derivation tree δ of $\vec{d}$ contains the label $\vec{c}$ and Γ is determined by δ.

Informally, the out-branching Γ of $\vec{c}$ outlines the "intended arrangement" of components of $\vec{c}$ in a (potential) resulting out-branching of G.

In our case we have got an out-branching Γ_1 of the aforementioned weak signature $\vec{c}_1$ (of (F_1, μ_1)) determined by the derivation tree δ_0. Let $(x, y) \in E(\Gamma_1)$ be its edge such that x is a copy of the weak shape (a, B), $i \in B$, and y is a copy of the weak shape (j, C), and that $\vec{c}_2$ results from $\vec{c}_1$ in the iteration of the "$r = \alpha_{i,j}(q)$" step (III) which picks the weak shapes (a, B) and (j, C) in $\vec{c}_1$. The digraph $\Gamma_1 - (x, y)$ has two weak components; X containing x and Y containing y. Let $F_1 = L_1 \cup L_1'$ be a partition of F_1 such that L_1 is formed by the out-trees corresponding to the vertices of X and L_1' is formed by those of Y, and, particularly, let $T_x \subseteq L_1, T_y \subseteq L_1'$ be the out-trees corresponding to x, y of Γ_1. Hence the root v_1 of T_y has F_1-label j and some potentially active vertex u_1 in T_x has F_1-label i,

We may as well assume that δ_1, its realization (F_1, μ_1) and x, y are chosen – subject to optimality in the previous criteria – such that they minimize the distance from the root of δ_1 to one of its nodes d_2 satisfying the following: In the annotated subforest (F_2, μ_2) induced from (F_1, μ_1) at the derivation node d_2 and containing u_1, there exists a vertex $u_2 \in V(F_2) \cap V(L_1)$ such that u_2 is active in (F_2, μ_2) and u_2 has the same F_2-label as u_1 (possibly $u_2 = u_1$). This leads to two cases to be considered:

i. The distance to our d_2 is zero. Then there is an active vertex $u_2 \in V(L_1)$ in (F_1, μ_1) of the F_1-label i, and so (u_2, v_1) is an edge of G.

ii. The distance to our d_2 is non-zero. Then, in particular, all active vertices of (F_1, μ_1) of the F_1-label i belong to L_1'.

Ad (i), we take the out-forest $F_1' = F_1 + (u_2, v_1) \subseteq G$. Let $\vec{c}_1{}'$ be the weak signature of the annotated out-forest (F_1', μ_1') where $\mu_1'(u_2) = \mu_1(u_2) - 1$ and $\mu_1'(x) = \mu_1(x)$ otherwise, and δ_1' the derivation tree of $\vec{c}_1{}'$ realizing (F_1', μ_1'). Let x' be the vertex of Γ_1 corresponding to the out-tree of F_1 containing u_2, let $\Gamma_1' = \Gamma_1 - (x, y) + (x', y)$, and Γ_1'' be obtained from Γ_1' by contracting (x', y). Clearly, Γ_1' is an out-branching of $\vec{c}_1$ and feasibility of Γ_1 naturally implies that also Γ_1' is feasible for t. Hence Γ_1'' is a feasible out-branching of $\vec{c}_1{}'$ for t. The derivation tree witnessing feasibility of Γ_1'' (in place of δ_0), its subtree δ_1' (in place of δ_1), and the annotated out-forest $(F_1' = F_1 + (u_2, v_1), \mu_1')$ contradict the optimality of our choice of $\vec{w}_0$ and δ_0 above. The proof is finished in this case (i).

Ad (ii), let the F_2-label of u_1 and u_2 be i'. Let d_3 be the parent node of d_2 in δ_1 and (F_3, μ_3) be induced from (F_1, μ_1) at d_3. By the optimality of our choice of d_2, the operation (cf. III) taking place at d_3 must be an iteration of $\alpha_{i',j'}$ adding an edge from u_2 (or u_2 would still be active in (F_3, μ_3)). Let $(u_2, v) \in E(F_3) \setminus E(F_2)$ be this added edge. Moreover, since u_1 is still potentially active in (F_1, μ_1), there exists a vertex $u_3 \in V(F_2) \cap V(L_1')$ of F_2-label i' active in (F_3, μ_3), and $(u_3, v) \in E(G)$.

Let $F_3' = F_2 + (u_3, v)$ and $\mu_3' = \mu_3$ except that $\mu_3'(u_2) = \mu_3(u_2) + 1$, $\mu_3'(u_3) = \mu_3(u_3) - 1$. The (F_3', μ_3') is an annotated out-forest in which u_2 is still active. Now, if u_3 is active in (F_1, μ_1), then we set $F_1' = F_1 - (u_2, v) + (u_3, v)$ and μ_1' accordingly. If u_3 is not active in (F_1, μ_1), then we pick any edge $(u_3, v') \in E(F_1) \setminus E(F_3)$ and subsequently define $F_1' = F_1 - (u_2, v) + (u_3, v) - (u_3, v') + (u_2, v')$ and $\mu_1' = \mu_1$. Again, it is routine to verify that F_1' is an out-forest in G in both cases. Let $\vec{c}_1{}'$ be the weak signature of new (F_1', μ_1') and δ_1' the derivation tree of $\vec{c}_1{}'$ realizing (F_1', μ_1').

Finally, we apply the computation run of the derivation tree δ_0 (starting up from the root of δ_1) onto the top of δ_1'. In this way we obtain an out-branching $\Gamma_1'$of $\vec{c}_1{}'$ that is an appropriate local modification of Γ_1. It follows from our choice of u_2, u_3 and their out-edges that Γ_1' is also feasible for t. Now we have an alternative optimal choice of $\vec{w}_0' \in \mathcal{V}$ and δ_0' which are witnessing feasibility of Γ_1', and of (F_1', μ_1') in place of (F_1, μ_1). This time, however, d_3 with δ_1' contradicts the optimality of our previous choice of d_2 (the distance from the root of δ_1' to d_3 is smaller by one).

This contradiction closes case (ii), and so the proof is finished. ◀

▶ **Question 14.** Seeing the complications in the proof of Theorem 13, one may naturally ask about a simpler solution of the problem. Say, cannot one come up with a better version of Definition 9 that, together with an appropriate modification of Lemma 10, would directly provide us with an XP algorithm? To be more formal, we ask whether there exists an equivalence relation $\sim$ on the set of annotated out-forests in a k-labelled graph H such that

- $\sim$ refines $\approx_H$ (Definition 8) for every particular H, and
- the set of nonempty classes of $\sim$ for particular H can be computed dynamically over a k-expression of H in XP time.

4 Edge-disjoint paths

▶ **Definition 15.** In the *Disjoint Paths problem*, an input is a graph (or digraph) G and k pairs of terminals $(s_1, t_1), \dots, (s_k, t_k)$, where $s_i, t_i \in V(G)$ for $1 \le i \le k$. The question is whether there exists a collection of k pairwise vertex-disjoint paths $P_1, \dots, P_k$ in G such that P_i connects s_i to t_i, $i = 1, \dots, k$.

The *Edge Disjoint Paths problem* is defined analogously with requiring the paths $P_1, \dots, P_k$ to be only pairwise edge-disjoint.

While the undirected Disjoint Paths variants are FPT solvable when parameterized simply by the number of paths (terminal pairs) [18], the directed case is NP-complete already for two paths in general. Hence it makes sense to look for suitable additional parameterizations of this problem, e.g. by clique-width. Note, on the other hand, that the Disjoint Paths problem with the number of paths k on the input is para-NP-complete for graphs of bounded clique-width [12], and the Edge Disjoint Paths problem with k on the input is para-NP-complete even for graphs of tree-width two [15].

▶ **Definition 16.** The *monadic second order logic* of one-sorted adjacency graphs, commonly abbreviated as MSO_1, has variables for graph vertices (say $x, y, z\dots$) and for vertex sets $(X, Y, Z\dots)$, common logic connectives and quantifiers, and a binary relational predicate *edge*. When dealing with directed graphs, we write *arc* instead of *edge*. Note that quantification over sets of edges is not possible (unlike in the more general MSO_2 language).

To give examples of MSO_1, we express that X is a dominating set in a graph G as $\delta(X) \equiv \forall y \notin X\, \exists z \in X\, edge(z, y)$, and that a digraph G is acyclic as $\alpha \equiv \forall X\, \exists y \in X\, \forall z \in X \neg\, arc(z, y)$. The MinLOB problem, on the other hand, is not expressible in MSO_1 (even with constant number of leaves) since neither the Hamiltonian Path is. Interestingly, the "dual" MaxLOB problem has an MSO_1 definition since, e.g. [10], a solution to MaxLOB is a complement to an out-connected dominating set in G.

Similarly, the (vertex) disjoint paths problem for a fixed k has a relatively easy description in MSO_1, e.g. [10]. For edge-disjoint paths with fixed k the situation is more complicated – the inability to handle sets of edges seems to prevent us from expressing that two paths (possibly sharing many vertices) are indeed edge disjoint. Yet, with a suitable trick we are able to express the existence of k directed pairwise edge-disjoint paths in MSO_1, and hence also to show membership in FPT when parameterized by clique-width or rank-width.

▶ **Theorem 17.** *Let G be a digraph, and $(s_1, t_1), \dots, (s_k, t_k)$ be pairs of terminals in G. There exists an MSO_1 formula π_k such that $G \models \pi_k(s_1, \dots, s_k, t_1, \dots, t_k)$ if, and only if, the corresponding* directed k edge-disjoint paths problem *in G has a solution.*

Proof. We start with an informal sketch of our approach. The initial idea is to focus on such collections of pairwise edge-disjoint s_i–t_i paths P_i, $1 \leq i \leq k$, in G that lexicographically minimize the length vector $\big(len(P_1), \dots, len(P_k)\big)$. So each P_i is an induced path in the subgraph $G - E(P_1 \cup \dots \cup P_{i-1})$. Then, by standard means, we "identify" each s_i–t_i path P_i with its vertex set X_i, and express the existence of P_i as the nonexistence of a separation between s_i, t_i inside X_i of $G - E(P_1 \cup \dots \cup P_{i-1})$. The difficult part of this solution is to specify the edges $E(P_j)$, $1 \leq j < i$. Formally, let

$$\begin{aligned} \varrho(x, y, Z, r) \equiv\ & \forall Y \big[(x \in Y \wedge y \notin Y) \rightarrow \\ & \exists z, z' \in Z \big(z \in Y \wedge z' \notin Y \wedge z \neq r \neq z' \wedge \operatorname{arc}(z, z')\big)\big] \end{aligned} \qquad (1)$$

be a formula stating that there exists a directed path from x to y on the vertices $Z \setminus \{r\}$ (note that $\{x, y\} \not\subseteq Z$ implies $G \not\models \varrho(x, y, Z, r)$), and put

$$\mu(s, t, Z, u, v) \;\equiv\; \varrho(s, u, Z, v) \wedge \varrho(v, t, Z, u) \qquad (2)$$

Claim. Let Z be a vertex subset of G such that $s, t \in Z$ and the subgraph $G[Z] \subseteq G$ induced on the vertices Z contains a directed s–t-path. Then the following three statements hold:

(i) If $G \models \mu(s, t, Z, u, v)$, then $\{s, t, u, v\} \subseteq Z$.
(ii) If $G \models \neg\mu(s, t, Z, u, v)$, then no s–t-path in $G[Z]$ may contain the edge (u, v).
(iii) Suppose Z is inclusion-minimal such that $G[Z]$ contains a s–t-path P. Then such P is unique and $E(P)$ is the set of those $(u, v) \in E(G)$ such that $G \models \mu(s, t, Z, u, v)$.

For (i) the proof follows directly from the definition of $\mu(s, t, Z, u, v)$. To see that (ii) also holds, it is enough to note that every edge (u, v) of every s–t-path in $G[Z]$ satisfies $G \models \mu(s, t, Z, u, v)$ by (1). Finally to prove (iii) let us suppose that $(u, v) \in E(G[Z]) \setminus E(P)$ (i.e., (u, v) points "backwards" on P due to minimality of Z). If in (2), for instance, $G \models \varrho(s, u, Z, v)$, then the corresponding s–u-path joined with the u–t-subpath of P would result in an s–t-path in $G[Z]$ avoiding v, a contradiction to minimality of Z. This finishes the proof of the claim.

Now, (iii) provides us with a criterion for identifying edges used by one particular s–t-path. To make use of it in a k path problem, we have to identify edges used by the first path P_1 in G, then edges used by P_2 in $G - E(P_1)$, then those used by P_3 in $G - E(P_1 \cup P_2)$, etc. For that we use the following trick which "replaces" the atomic predicate *arc* in (1) with appropriate recursively defined (3) formulas α_j where $j = 1, \dots, k$. For simplicity, we write $\widehat{s_j}$ as a shortcut for the list $s_1, s_2 \dots, s_j$, and analogically for $\widehat{t_j}, \widehat{X_j}$.

$$\begin{aligned}\alpha_1(u,v) \;&\equiv\; \mathit{arc}(u,v)\,, \\ \alpha_{j+1}(u,v,\widehat{s_j},\widehat{t_j},\widehat{X_j}) \;&\equiv\; \alpha_j(u,v,\widehat{s_{j-1}},\widehat{t_{j-1}},\widehat{X_{j-1}}) \wedge \neg\mu_j(s_j,t_j,X_j,u,v,\widehat{s_{j-1}},\widehat{t_{j-1}},\widehat{X_{j-1}})\end{aligned} \tag{3}$$

where $\mu_j \equiv \varrho_j(s_j,u,X_j,v,\widehat{s_{j-1}},\widehat{t_{j-1}},\widehat{X_{j-1}}) \wedge \varrho_j(v,t_j,X_j,u,\widehat{s_{j-1}},\widehat{t_{j-1}},\widehat{X_{j-1}})$ analogically to (2), and ϱ_j is replacing the *arc* predicate in ϱ (1) simply as follows

$$\begin{aligned}\varrho_j(x,y,Z,r,\widehat{s_{j-1}},\widehat{t_{j-1}},\widehat{X_{j-1}}) \;\equiv\; &\forall Y \left[(x \in Y \wedge y \notin Y) \rightarrow \right. \\ &\left.\exists z,z' \in Z \left(z \in Y \wedge z' \notin Y \wedge z \neq r \neq z' \wedge \alpha_j(z,z',\widehat{s_{j-1}},\widehat{t_{j-1}},\widehat{X_{j-1}})\right)\right].\end{aligned} \tag{4}$$

Writing just ϱ_j' in place of previous ϱ_j "without r" (4), we obtain the solution

$$\begin{aligned}\pi_k(\widehat{s_k},\widehat{t_k}) \;\equiv\; \exists \widehat{X_k}\; &\varrho_1'(s_1,t_1,X_1) \wedge \varrho_2'(s_2,t_2,X_2,\widehat{s_1},\widehat{t_1},\widehat{X_1}) \wedge \\ &\cdots \wedge \varrho_k'(s_k,t_k,X_k,\widehat{s_{k-1}},\widehat{t_{k-1}},\widehat{X_{k-1}})\,.\end{aligned}$$

It remains to prove that $G \models \pi_k(\widehat{s_k},\widehat{t_k})$ if, and only if, there exist k edge-disjoint s_i–t_i paths in G where $i = 1, \dots, k$. In one direction, suppose a particular choice of the vertex sets $\widehat{X_k}$ satisfying π_k on G. According to (3) and (4) this assumption means that, for each $i = 1, \dots, k$ by induction, there exists a directed s_i–t_i path P_i on the vertices X_i such that P_i completely avoids (ii) edges potentially used by the paths $P_1, \dots, P_{i-1}$. Hence such $P_1, \dots, P_k$ are pairwise edge-disjoint in G.

Conversely, among all collections of k pairwise edge-disjoint s_i–t_i paths P_i in G, we select one lexicographically minimizing the vector $\big(len(P_1), \dots, len(P_k)\big)$. Then, clearly, each $X_i = V(P_i)$ for $i = 1, \dots, k$ is inclusion-minimal inducing an s_i–t_i path in $G - E(P_1 \cup \dots \cup P_{i-1})$, and so the claim (iii) applies here. Hence, by induction on i, we conclude from (3) that $G \models \alpha_{i+1}(u,v,\widehat{s_i},\widehat{t_i},\widehat{X_i})$ iff $(u,v) \in E(G) \setminus E(P_1 \cup \dots \cup P_i)$. And since $P_{i+1} \subseteq G - E(P_1 \cup \dots \cup P_i)$, it follows from (4) that our selected sets $X_1, \dots, X_k$ satisfy $\pi_k(\widehat{s_k},\widehat{t_k})$ on G. ◀

In connection with [3][1] we finally obtain:

▶ **Corollary 18.** *Both the undirected and directed edge-disjoint paths problems with fixed k have a linear FPT algorithm on simple (di)graphs of bounded clique-width.*

▶ **Question 19.** Notice that the MSO$_1$ formula π_k constructed in the proof of Theorem 17 has quantifier alternation depth growing with k. Therefore the worst-case runtime estimate of Corollary 18 coming from [3] has a tower-exponential dependency on the parameter k. The question thus is whether an MSO$_1$ description of the k edge-disjoint paths problem is possible with fixed quantifier alternation depth. (An ad-hoc estimate of the Myhill–Nerode congruence classes of the problem suggests this might be true.)

[1] Note that [3] considered only undirected graphs, but the same results also hold for digraphs, cf. [14, 8].

References

1 J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications.* Springer Monographs in Mathematics. Springer, second edition, 2009.

2 D. Corneil and U. Rotics. On the relationship between cliquewidth and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005.

3 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.

4 B. Courcelle and S. Olariu. Upper bounds to the clique width of graphs. *Discrete Appl. Math.*, 101(1-3):77–114, 2000.

5 P. Dankelmann, G. Gutin, and E. Kim. On complexity of minimum leaf out-branching problem. *Discrete Appl. Math.*, 157(13):3000–3004, 2009.

6 W. Espelage, F. Gurski, and E. Wanke. How to solve NP-hard graph problems on clique-width bounded graphs in polynomial time. In *WG'01*, volume 2204 of *LNCS*, pages 117–128. Springer, 2001.

7 F. Fomin, P. Golovach, D. Lokshtanov, and S. Saurab. Clique-width: On the price of generality. In *SODA'09*, pages 825–834. SIAM, 2009.

8 R. Ganian and P. Hliněný. On parse trees and Myhill–Nerode–type tools for handling graphs of bounded rank-width. *Discrete Appl. Math.*, 158:851–867, 2010.

9 R. Ganian, P. Hliněný, J. Kneis, A. Langer, J. Obdržálek, and P. Rossmanith. On digraph width measures in parametrized algorithmics. In *IWPEC'09*, volume 5917 of *LNCS*, pages 161–172. Springer, 2009.

10 R. Ganian, P. Hliněný, J. Kneis, A. Langer, J. Obdržálek, and P. Rossmanith. Digraph width measures in parametrized algorithmics. Submitted. Available from: `http://www.fi.muni.cz/~hlineny/Research/papers/kenny-full.pdf`, 2010.

11 R. Ganian, P. Hliněný, and J. Obdržálek. Unified approach to polynomial algorithms on graphs of bounded (bi-)rank-width. Submitted, 2010. 29 p.

12 F. Gurski and E. Wanke. Vertex disjoint paths on clique-width bounded graphs. *Theoret. Comput. Sci.*, 359(1-3):188–199, 2006.

13 P. Hliněný and S. Oum. Finding branch-decomposition and rank-decomposition. *SIAM J. Comput.*, 38:1012–1032, 2008.

14 M. Kanté. The rank-width of directed graphs. arXiv:0709.1433v3, March 2008.

15 T. Nishizeki, J. Vygen, and X. Zhou. The edge-disjoint path problem is NP-complete for series-parallel graphs. *Discrete Appl. Math.*, 115(1-3):177–186, 2001.

16 S. Oum and P. D. Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

17 N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

18 N. Robertson and P. D. Seymour. Graph minors. XIII: The disjoint paths problem. *J. Comb. Theory Ser. B*, 63(1):65–110, 1995.

19 G. Salamon and G. Wiener. On finding spanning trees with few leaves. *Inform. Process. Lett.*, 105(5):164–169, 2008.

From Pathwidth to Connected Pathwidth

Dariusz Dereniowski*[1]

[1] Department of Algorithms and System Modeling, Gdańsk University of Technology, Narutowicza 11/12, 80-233 Gdańsk, Poland
deren@eti.pg.gda.pl

Abstract

It is proven that the connected pathwidth of any graph G is at most $2 \cdot \mathrm{pw}(G) + 1$, where $\mathrm{pw}(G)$ is the pathwidth of G. The method is constructive, i.e. it yields an efficient algorithm that for a given path decomposition of width k computes a connected path decomposition of width at most $2k + 1$. The running time of the algorithm is $O(dk^2)$, where d is the number of 'bags' in the input path decomposition.

The motivation for studying connected path decompositions comes from the connection between the pathwidth and some graph searching games. One of the advantages of the above bound for connected pathwidth is an inequality $\mathtt{cs}(G) \leq 2\mathtt{s}(G) + 3$, where $\mathtt{cs}(G)$ is the connected search number of a graph G and $\mathtt{s}(G)$ is its search number, which holds for any graph G. Moreover, the algorithm presented in this work can be used to convert efficiently a given search strategy using k searchers into a connected one using $2k + 3$ searchers and starting at arbitrary homebase.

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

Keywords and phrases connected pathwidth, connected searching, fugitive search games, graph searching, pathwidth

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.416

1 Introduction

The notions of pathwidth and treewidth are receiving increasing interest since the series of Graph Minor articles by Robertson and Seymour. The importance of those parameters is due to their numerous practical applications, connections with several graph parameters and usefulness in designing graph algorithms. Informally speaking, the *pathwidth* of a graph G, denoted by $\mathrm{pw}(G)$, says how closely G is related to a path. Moreover, a path decomposition captures the linear path-like structure of G. (For a definition see Section 2.)

Here we briefly describe a graph searching game that is the main motivation for the results presented in this paper. A team of k searchers is given and the goal is to find an invisible and fast fugitive located in a given graph G. The fugitive has also the complete knowledge about the graph and about the strategy used by the searchers, and therefore he will avoid being captured as long as possible. The fugitive is captured when a searcher reaches his location. In this setting the game is equivalent to the problem of clearing all edges of a graph that is initially entirely contaminated. There are two main types of this graph searching problem. In the *node searching* two moves are allowed: placing a searcher on a vertex and removing a searcher from a vertex. An edge becomes clear whenever both of its

* Research partially supported by MNiSW grant N N206 379337 (2009-2011).

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 416–427
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

endpoints are simultaneously occupied by searchers. In the *edge searching* we have, besides to the two mentioned moves, a move of sliding a searcher along an edge. In this model an edge $\{u, v\}$ becomes clear if a searcher slides from u to v and either all the other edges incident to u have been previously cleared or another searcher occupies u. In both cases the goal is to find a search strategy (a sequence of moves of the searchers) that clears all the edges of G. The *node* (*edge*) *search number* of G, denoted by $\mathtt{ns}(G)$ ($\mathtt{s}(G)$, respectively), equals the minimum number of searchers sufficient to construct a node (edge, respectively) search strategy. An important property is that $\mathrm{pw}(G) = \mathtt{ns}(G) - 1$ for any graph G [13, 14, 15, 18]. The edge searching problem is closely related to node searching, i.e. $|\mathtt{s}(G) - \mathtt{ns}(G)| \leq 1$ [4], and consequently to pathwidth, $\mathrm{pw}(G) \leq \mathtt{s}(G) \leq \mathrm{pw}(G) + 2$.

In this work we are interested in special types of path decompositions called connected path decompositions. The motivation comes from the need of creating connected search strategies. An edge search strategy is *connected* if the subgraph of G that is clear is always connected. The minimum number of searchers that guarantees the capture of the fugitive by a connected (edge) search strategy, denoted by $\mathtt{cs}(G)$, is the *connected search number* of G. This model of graph searching receives recently a growing interest, because in many applications the connectedness is an requirement.

Related work. There are several results that give a relation between the connected and the 'classical' search numbers of a graph. Fomin et al. proved in [7] that the connected search number of an n-node graph of branchwidth b is bounded by $O(b \log n)$ and this bound is tight. One of the implications of this result is that $\mathtt{cs}(G) = O(\log n)\mathrm{pw}(G)$. Nisse proved in [19] that $\mathtt{cs}(G) \leq (\mathrm{tw}(G)+2)(2\mathtt{s}(G)-1)$ for any chordal graph G. Barrière et al. obtained in [2] a constant upper bound for trees, namely for each tree T, $\mathtt{cs}(T)/\mathtt{s}(T) \leq 2$. On the other hand, there exists an infinite family of graphs G_k such that $\mathtt{cs}(G_k)/\mathtt{s}(G_k)$ approaches 2 when k goes to infinity [3].

Fraigniaud and Nisse presented in [9] a $O(nk^3)$-time algorithm that takes a width k tree decomposition of a graph and returns a connected tree decomposition of the same width. (For definition of treewidth see e.g. [5, 21].) Therefore, $\mathrm{tw}(G) = \mathrm{ctw}(G)$ for any graph G. This method cannot be applied for proving the same result for connected path decompositions, because the decomposition that the algorithm in [9] constructs is not, in general, a path decomposition even when a path decomposition is given as an input. That result also yields an upper bound of $\mathtt{cs}(G) \leq (\log n + 1)\mathtt{s}(G)$ for any graph G. The problems of computing the pathwidth (the search number) and the connected pathwidth (the connected search number) are NP-hard, also for several special classes of graphs, see e.g. [6, 11, 12, 16, 17, 20].

This work. This paper presents an efficient algorithm that takes a (connected) graph G and its path decomposition $\mathcal{P} = (X_1, \ldots, X_d)$ of width k as an input and finds in time $O(dk^2)$ a connected path decomposition $\mathcal{C} = (Z_1, \ldots, Z_m)$ of width at most $2k + 1$, where $m \leq kd$. This solves an open problem stated in several papers, e.g. in [1, 2, 3, 7, 8, 9, 10, 22], since it implies that for any graph G, $\mathrm{cpw}(G) \leq 2\mathrm{pw}(G) + 1$, and improves previously known estimations [7, 19]. The path decomposition $\mathcal{C}$ can be turned into a monotone connected search strategy using at most $2k + 3$ searchers. Thus, in terms of the graph searching terminology, the bound immediately implies that $\mathtt{mcs}(G) \leq \mathrm{cpw}(G) + 2 \leq 2\mathrm{pw}(G) + 3 \leq 2\mathtt{s}(G) + 3$, where $\mathtt{mcs}(G)$ is the monotone connected search number of G. (A search strategy is *monotone* if the fugitive cannot reach a previously cleared edge.) Since $\mathtt{cs}(G) \leq \mathtt{mcs}(G)$, the bound can be restated for the connected search number of a graph, $\mathtt{cs}(G) \leq 2\mathtt{s}(G)+3$. Moreover, the factor 2 in the bound is tight [3]. The bound finds also applications in designing approximation algorithms, for it implies that the pathwidth and the connected pathwidth (the search number, the connected search number, and some other search numbers not mentioned here, e.g. the internal search number) are within a constant factor of each other.

2 Preliminaries and basic definitions

Given a simple graph $G = (V(G), E(G))$ and its subset of vertices $X \subseteq V(G)$, the subgraph of G *induced* by X is $G[X] = (X, \{\{u, v\} \in E(G) \colon u, v \in X\})$. For a simple (not necessary connected) graph G, H is its connected component if H is connected, that is, there exists a path in H between each pair of vertices, and each proper supergraph of H is not a subgraph of G. For $X \subseteq V(G)$ let $N_G(X) = \{u \in V(G) \setminus X \colon \{u, x\} \in E(G)$ for some $x \in X\}$.

▶ **Definition 1.** A *path decomposition* of a simple graph $G = (V(G), E(G))$ is a sequence $\mathcal{P} = (X_1, \ldots, X_d)$, where $X_i \subseteq V(G)$ for each $i = 1, \ldots, d$, and

- ○ $\bigcup_{i=1,\ldots,d} X_i = V(G)$,
- ○ for each $\{u, v\} \in E(G)$ there exists $i \in \{1, \ldots, d\}$ such that $u, v \in X_i$,
- ○ for each i, j, k, $1 \leq i \leq j \leq k \leq d$ it holds $X_i \cap X_k \subseteq X_j$.

The *width* of the path decomposition $\mathcal{P}$ is $\mathrm{width}(\mathcal{P}) = \max_{i=1,\ldots,d} |X_i| - 1$. The *pathwidth* of G, $\mathrm{pw}(G)$, is the minimum width over all path decompositions of G.

A path decomposition $\mathcal{P}$ is *connected* if $G[X_1 \cup \cdots \cup X_i]$ is connected for each $i = 1, \ldots, d$. Then, $\mathrm{cpw}(G)$ denotes the minimum width over all connected path decompositions of G.

▶ **Definition 2.** Given a graph G and its path decomposition $\mathcal{P} = (X_1, \ldots, X_d)$, a node-weighted graph $\mathcal{G} = (V(\mathcal{G}), E(\mathcal{G}), \omega)$ *derived* from G and $\mathcal{P}$ is the graph with vertex set $V(\mathcal{G}) = V_1 \cup \cdots \cup V_d$, where $V_i = \{v_i(H) \colon H$ is a connected component of $G[X_i]\}$, $i = 1, \ldots, d$, and edge set $E(\mathcal{G}) = \{\{v_i(H), v_{i+1}(H')\} \colon v_i(H) \in V_i, v_{i+1}(H') \in V_{i+1}, i \in \{1, \ldots, d-1\}$, and $V(H) \cap V(H') \neq \emptyset\}$. The weight of a vertex $v_i(H) \in V(\mathcal{G})$, $i \in \{1, \ldots, d\}$, is $\omega(v_i(H)) = |V(H)|$. The *width* of $\mathcal{G}$, denoted by $\mathrm{width}(\mathcal{G})$, equals $\mathrm{width}(\mathcal{P}) + 1$.

In the following we omit a subgraph H of G and the index $i \in \{1, \ldots, d\}$ whenever they are not important when referring to a vertex of $\mathcal{G}$ and we write v instead of $v_i(H)$. For brevity, $\omega(X) = \sum_{x \in X} \omega(x)$ for any subset $X \subseteq V(\mathcal{G})$.

Figures 1(a) and 1(b) present a graph G and its path decomposition $\mathcal{P}$, respectively, where the subgraph structure in each bag X_i is also given. Figure 1(c) depicts the derived graph $\mathcal{G}$. Note that $\mathcal{P}$ is not connected: the subgraphs $G[X_1 \cup \cdots \cup X_i]$ are not connected for $i = 2, 3, 4$. Let $C \subseteq V(\mathcal{G})$. The *border* $\delta(C)$ of the set C is its subset consisting of all the vertices $v \in C$ such that there exists $u \in V(\mathcal{G}) \setminus C$ adjacent to v in $\mathcal{G}$, i.e. $\delta(C) = N_{\mathcal{G}}(V(\mathcal{G}) \setminus C)$.

Given a set $X \subseteq V(\mathcal{G})$, $X \neq \emptyset$, we define the *left* (*right*) *extremity* of X as $l(X) = \min\{i \colon V_i \cap X \neq \emptyset\}$ ($r(X) = \max\{i \colon V_i \cap X \neq \emptyset\}$, respectively).

A path P in $\mathcal{G}$ is *progressive* if $|V(P) \cap V_i| \leq 1$ for each $i = 1, \ldots, d$.

▶ **Definition 3.** Given $\mathcal{G}$, $C \subseteq V(\mathcal{G})$ and $X \subseteq \delta(C)$, a *left* (*right*) *branch* $\mathcal{B}_{\mathrm{L}}(C, X, i)$, where $1 \leq i \leq r(X)$ (respectively $\mathcal{B}_{\mathrm{R}}(C, X, i)$, where $l(X) \leq i \leq d$) is the subgraph of $\mathcal{G}$ induced by the vertices in X and by the vertices of all progressive paths contained in $(V(\mathcal{G}) \setminus C) \cup X$ and connecting $x \in X \cap V_j$ and $v \in V_k \setminus C$, where $i \leq k \leq j$ ($j \leq k \leq i$, respectively).

We sometimes write $\mathcal{B}$ to refer to a branch whenever its 'direction' or C, X, i are clear from the context. A branch $\mathcal{B} = \mathcal{B}_{\mathrm{L}}(C, X, i)$, $i \leq r(X)$, ($\mathcal{B} = \mathcal{B}_{\mathrm{R}}(C, X, i)$, $i \geq l(X)$) is *continuous* if $V_j \cap V(\mathcal{B}) \neq \emptyset$ for each $j = i, \ldots, r(X)$ ($j = l(X), \ldots, i$, respectively). A vertex v of $\mathcal{B}$ is *external* if $N_{\mathcal{G}}(v) \not\subseteq C \cup V(\mathcal{B})$. The branch $\mathcal{B}$ is *proper* if it has no external vertices in

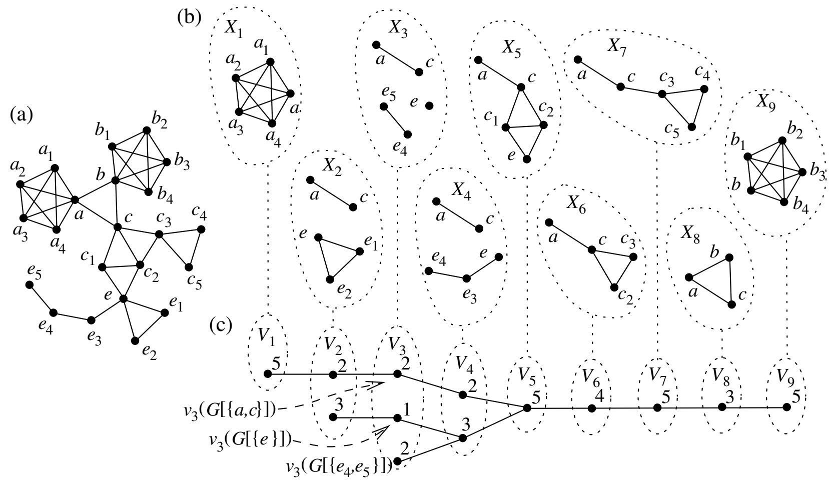


Figure 1 (a) a graph G; (b) a path decomposition $\mathcal{P}$ of G; (c) the graph $\mathcal{G}$ derived from G and $\mathcal{P}$

$V_{i+1} \cup \cdots \cup V_{r(X)-1}$ ($V_{l(X)+1} \cup \cdots \cup V_{i-1}$, respectively), while $\mathcal{B}$ is *maximal* if it is continuous, proper and $\mathcal{B}_{\mathrm{L}}(C, X, i-1)$ ($\mathcal{B}_{\mathrm{R}}(C, X, i+1)$, respectively) is not a proper branch. An integer j is a *cut* of the branch $\mathcal{B}$ if $i \leq j \leq r(X)$ ($l(X) \leq j \leq i$, respectively). The *weight* of the cut j of $\mathcal{B}$ is $\omega((V(\mathcal{B}) \cap V_j) \cup (X \cap Y))$, where $Y = V_{l(X)} \cup \cdots \cup V_{j-1}$ for a left branch, and $Y = V_{i+1} \cup \cdots \cup V_{r(X)}$ for a right branch. A cut of minimum weight is a *bottleneck* of $\mathcal{B}$.

Figure 2 illustrates the above definitions. (In all cases the branch is distinguished by the dark area.) Let $X = \{x_1, x_2, x_3\}$ be a subset of $\delta(C)$. Figure 2(a) gives $\mathcal{B}_{\mathrm{L}}(C, X, i_2)$ and this branch is continuous and proper, but not maximal for each i_2, $i_3 < i_2 \leq i_1$. The branch $\mathcal{B}_{\mathrm{L}}(C, X, i_3)$ (see Figure 2(b)) is maximal (thus continuous and proper), which follows from the fact that any branch $\mathcal{B}_{\mathrm{L}}(C, X, i_4)$, where $i_4 < i_3$ is not proper, because it contains an external vertex u_2, as shown in Figure 2(c). Note that the vertices of $\mathcal{G}_1$ and $\mathcal{G}_2$ (except for u_1 and u_2) do not belong to any branch $\mathcal{B}_{\mathrm{L}}(C, X, i)$, because they are not connected by progressive paths to x_2 or x_3. Figure 2(d) depicts a branch $\mathcal{B}_{\mathrm{L}}(C, X, i_7)$ that is not continuous for any $i_7 < i_5$, because $V_{i_5-1} \cap V(\mathcal{B}_{\mathrm{L}}(C, X, i_7)) = \emptyset$. In our algorithm we ensure that each branch we use is continuous and proper.

Given C, X and i, a left (respectively right) branch $\mathcal{B}$ can be calculated efficiently as follows. Initially $\mathcal{B}$ satisfies $V(\mathcal{B}) = X$. We start with $j = r(X)$ ($j = l(X)$, resp.) and we add to the vertex set of the branch all vertices in $V_{j-1} \setminus C$ ($V_{j+1} \setminus C$, resp.) that have a neighbor in $V_j \cap V(\mathcal{B})$. Then, we decrement (increment, resp.) j and repeat this step. The computation stops when $j < i$ ($j > i$, respectively).

3 The algorithm

The algorithm `CP` (*Connected Pathwidth*) for finding a connected path decomposition of a graph G takes G, a vertex v of G, and a path decomposition $\mathcal{P}$ of G as an input. The vertex v is guaranteed to belong to the first bag of the resulting connected path decomposition. This flexibility is provided due to the potential application of this algorithm: in graph searching games the bags of path decompositions correspond to the vertices occupied by the searchers while the search proceeds; in this way the selected vertex v can be the first one that becomes guarded in a connected search strategy and it is called the *homebase*. The first step performed by `CP` is the construction of the derived graph $\mathcal{G}$ and in the subsequent

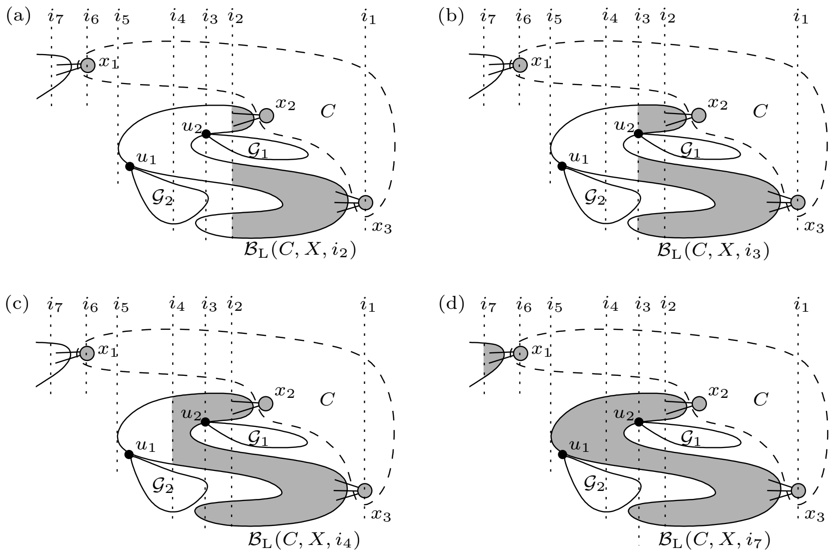

(a)
(b)
(c)
(d)
i_7
i_6
i_5
i_4
i_3
i_2
i_1
x_1
x_2
x_3
C
u_1
u_2
$\mathcal{G}_1$
$\mathcal{G}_2$
$\mathcal{B}_{\mathrm{L}}(C, X, i_2)$
$\mathcal{B}_{\mathrm{L}}(C, X, i_3)$
$\mathcal{B}_{\mathrm{L}}(C, X, i_4)$
$\mathcal{B}_{\mathrm{L}}(C, X, i_7)$

Figure 2 $\mathcal{G}$ with distinguished vertex sets $X = \{x_1, x_2, x_3\}$ and C, and the corresponding branches that are: (a) continuous and proper but not maximal; (b) maximal; (c) continuous but not proper (thus not maximal); (d) not continuous nor proper (thus not maximal)

steps the algorithm works on $\mathcal{G}$. (Also, most parts of our analysis use $\mathcal{G}$ rather than G.) The algorithm computes a sequence of sets $C_j \subseteq V(\mathcal{G})$, $j = 1, \ldots, m$, called *expansions*. The expansion C_1 consists of v and one of its neighbors, and $C_m = V(\mathcal{G})$ at the end of the execution of `CP`. Moreover, $C_j \subseteq C_{j+1}$ for each $j = 1, \ldots, m-1$. Informally speaking, C_{j+1} is obtained from C_j by adding to C_j some vertices from $N_{\mathcal{G}}(C_j)$. This guarantees that the final path decomposition obtained from $\delta(C_1), \ldots, \delta(C_m)$ is valid and is connected, as proved in Lemma 7. On the other hand, the particular vertices in $N_{\mathcal{G}}(C_j)$, used to obtain C_{j+1}, are selected in a way to guarantee that $\omega(\delta(C_j))$ is bounded by $2 \cdot \mathrm{width}(\mathcal{G})$ for each $j = 1, \ldots, m$. By construction, $\omega(\delta(C_j))$ is the size of the corresponding jth bag in the resulting connected path decomposition.

In this section we give the statement of the algorithm and we prove that it computes a connected path decomposition $\mathcal{C}$. Then, in Section 4 we analyze the width of $\mathcal{C}$. Due to the space limitations, the proofs of several results (marked with $\triangle$) are omitted.

The algorithm computes for each expansion C_j two sets called the *left* and *right borders* of C_j, denoted by $\delta_{\mathrm{L}}(C_j)$ and $\delta_{\mathrm{R}}(C_j)$, respectively. It is guaranteed that $\delta_{\mathrm{L}}(C_j) \cup \delta_{\mathrm{R}}(C_j) = \delta(C_j)$ for each $j = 1, \ldots, m$ (see Lemma 6). As it is proven later, the left and right borders are special types of partitions of $\delta(C_j)$. In particular, there exists an integer $i \in \{1, \ldots, d\}$ such that the left border $\delta_{\mathrm{L}}(C_j)$ is contained in $V_1 \cup \cdots V_i$ and the right border $\delta_{\mathrm{R}}(C_j)$ is a subset of $V_{i+1} \cup \cdots \cup V_d$. For brevity let in the following $l(\delta_{\mathrm{L}}(C_j)) = r(\delta_{\mathrm{L}}(C_j)) = 0$ if $\delta_{\mathrm{L}}(C_j) = \emptyset$ and $l(\delta_{\mathrm{R}}(C_j)) = r(\delta_{\mathrm{R}}(C_j)) = d$ if $\delta_{\mathrm{R}}(C_j) = \emptyset$, where C_j is any expansion.

We start by describing a subroutine `EE` (*Extend Expansion*) that is used by the main procedure `CP` given below. The input to `EE` consists of two integers i and k, $i, k \in \{1, \ldots, d\}$. Informally speaking, the procedure adds, in its subsequent iterations, to the current expansion C_m each vertex in V_j that is connected by a progressive path to a vertex in $V_{j'} \cap \delta(C_m)$ for each $j = i$ to k and for some j', $i \leq j' \leq j$ if $i < k$, and for each $j = i$ down to k, $j \leq j' \leq i$ if $i > k$, which we formally prove in Lemma 4 below. All the 'intermediate' expansions are recorded as they will give us the corresponding bags in the final path decomposition. The procedures `EE`(i, k) and `CP`$(G, \mathcal{P})$ are as follows.

```
Procedure EE (Extend Expansion)
Input: integers $i$ and $k$. ($\mathcal{G}$, $m$, $C_m$ are used as global variables)
    while $k \neq i$ do
        if $k < i$ then
            EL: Increment $m$, decrement $i$ and set:
                $C_m = C_{m-1} \cup (V_i \cap N_{\mathcal{G}}(C_{m-1}))$,
                $\delta_{\mathrm{L}}(C_m) = (\delta_{\mathrm{L}}(C_{m-1}) \cup V_i) \cap \delta(C_m)$,
                $\delta_{\mathrm{R}}(C_m) = \delta_{\mathrm{R}}(C_{m-1}) \cap \delta(C_m)$.
        else ($k > i$)
            ER: Increment $m$, increment $i$ and set:
                $C_m = C_{m-1} \cup (V_i \cap N_{\mathcal{G}}(C_{m-1}))$,
                $\delta_{\mathrm{R}}(C_m) = (\delta_{\mathrm{R}}(C_{m-1}) \cup V_i) \cap \delta(C_m)$,
                $\delta_{\mathrm{L}}(C_m) = \delta_{\mathrm{L}}(C_{m-1}) \cap \delta(C_m)$.
        end if
    end while.
end procedure EE.
```

Algorithm `CP` (*Connected Pathwidth*)
Input: a simple graph G, a path decomposition $\mathcal{P}$ of G, and a vertex $v \in V(G)$.
Output: a connected path decomposition $\mathcal{C}$ of G.

(*Initialization.*)

I.1: Use G and $\mathcal{P}$ to calculate the derived graph $\mathcal{G}$. Let v be any vertex of $\mathcal{G}$. Let $C_1 = \{x, y\}$, where $v \in C_1$, x, y are adjacent in $\mathcal{G}$, and $x \in V_i$, $y \in V_{i+1}$ for some $i \in \{1, \ldots, d-1\}$. Let $m = 1$.

I.2: If $x \in \delta(C_1)$, then set $\delta_{\mathrm{L}}(C_1) = \{x\}$, compute the maximal left branch $\mathcal{B}_{\mathrm{L}}(C_1, \delta_{\mathrm{L}}(C_1), a_0)$ with a bottleneck a_0' ($a_0' \geq a_0$) and with no external vertices in V_i and call `EE`(i, a_0'); otherwise $\delta_{\mathrm{L}}(C_1) = \emptyset$.

I.3: If $y \in \delta(C_1)$, then set $\delta_{\mathrm{R}}(C_1) = \{y\}$, compute the maximal right branch $\mathcal{B}_{\mathrm{R}}(C_1, \delta_{\mathrm{R}}(C_1), b_0)$ with a bottleneck b_0' ($b_0' \leq b_0$) and with no external vertices in V_{i+1} and call `EE`$(i+1, b_0')$; otherwise $\delta_{\mathrm{R}}(C_1) = \emptyset$.

```
(Main loop.)
while $C_m \neq V(\mathcal{G})$ do
    if $\omega(\delta_{\mathrm{L}}(C_m)) > \omega(\delta_{\mathrm{R}}(C_m))$ then
        L.1: Compute the maximal left branch $\mathcal{B}_1 = \mathcal{B}_{\mathrm{L}}(C_m, \delta_{\mathrm{L}}(C_m), k_1)$. If $\mathcal{B}_1$
             has no external vertex in $V_i$, $i = r(\delta_{\mathrm{L}}(C_m))$, then call EE$(r(\delta_{\mathrm{L}}(C_m)), k_1)$,
             otherwise let $k_1 = r(\delta_{\mathrm{L}}(C_m))$.
        L.2: Compute the maximal right branch $\mathcal{B}_2 = \mathcal{B}_{\mathrm{R}}(C_m, \delta_{\mathrm{R}}(C_m) \cup (V_{k_1} \cap$
             $\delta_{\mathrm{L}}(C_m)), k_2)$. Let $k_2'$ be its minimum weight cut such that $k_2' > k_1$. Call
             EE$(k_1, k_2')$.
        L.3: If $r(\delta_{\mathrm{L}}(C_m)) = k_1$, then compute the maximal left branch $\mathcal{B}_3 =$
             $\mathcal{B}_{\mathrm{L}}(C_m, \delta_{\mathrm{L}}(C_m), k_3)$ with bottleneck $k_3'$ and call EE$(k_1, k_3')$.
    else ($\omega(\delta_{\mathrm{L}}(C_m)) \leq \omega(\delta_{\mathrm{R}}(C_m))$)
        R.1: Compute the maximal right branch $\mathcal{B}_1 = \mathcal{B}_{\mathrm{R}}(C_m, \delta_{\mathrm{R}}(C_m), k_1)$. If $\mathcal{B}_1$
             has no external vertex in $V_i$, $i = l(\delta_{\mathrm{R}}(C_m))$, then call EE$(l(\delta_{\mathrm{R}}(C_m)), k_1)$,
             otherwise let $k_1 = l(\delta_{\mathrm{R}}(C_m))$.
```

```
            R.2: Compute the maximal left branch $\mathcal{B}_2 = \mathcal{B}_{\mathrm{L}}(C_m, \delta_{\mathrm{L}}(C_m) \cup (V_{k_1} \cap \delta_{\mathrm{R}}(C_m)), k_2)$. Let $k_2'$ be its minimum weight cut such that $k_2' < k_1$.
                 Call EE$(k_1, k_2')$.
            R.3: If $l(\delta_{\mathrm{R}}(C_m)) = k_1$, then compute the maximal right branch $\mathcal{B}_3 = \mathcal{B}_{\mathrm{R}}(C_m, \delta_{\mathrm{R}}(C_m), k_3)$ with bottleneck $k_3'$ and call EE$(k_1, k_3')$.
        end if
    end while.
    Let $Z_j = \bigcup_{v_k(H)\in\delta(C_j)} V(H)$ for each $j = 1, \ldots, m$. Return $\mathcal{C} = (Z_1, \ldots, Z_m)$.
end procedure CP.
```

First we briefly discuss the initialization stage of CP. In Step I.1 an expansion C_1 is constructed in such a way that it contains any two adjacent vertices (the adjacency guarantees the connectedness of the final path decomposition) such that one of them is the input vertex v. (W.l.o.g. v has a neighbor in G, because otherwise $\mathcal{G}$ contains a single vertex and therefore $\mathcal{P}$ is connected.) Steps I.2 and I.3 are symmetric. In Step I.2 (I.3) the algorithm finds the maximal left (right) branch 'emanating' from x (resp. y) provided that the vertex belongs to the border of C_1, otherwise the left (right, respectively) border is empty.

In the following, one *iteration* of CP or EE means one iteration of the 'while' loop in the corresponding procedure. Thus, in the case of CP, one iteration reduces to executing Steps L.1-L.3 or R.1-R.3 within the 'if' statement, while in the procedure EE one iteration results in executing the instructions in Step EL or in Step ER. We use the symbols $\mathcal{B}_1, \mathcal{B}_2, \mathcal{B}_3$, a_0, a_0', b_0, b_0', k_i, k_i' to refer to the variables used in CP, where in the case of k_1 we refer to its value at the end of Step L.1 or Step R.1. In what follows we denote for brevity $\mathcal{B}_2' = \mathcal{B}_{\mathrm{R}}(C_m, \delta_{\mathrm{R}}(C_m) \cup (V_{k_1} \cap \delta_{\mathrm{L}}(C_m)), k_2')$ and $\mathcal{B}_3' = \mathcal{B}_{\mathrm{L}}(C_m, \delta_{\mathrm{L}}(C_m), k_3')$ if Steps L.1-L.3 have been executed in this particular iteration of CP, and $\mathcal{B}_2' = \mathcal{B}_{\mathrm{L}}(C_m, \delta_{\mathrm{L}}(C_m) \cup (V_{k_1} \cap \delta_{\mathrm{R}}(C_m)), k_2')$, $\mathcal{B}_3' = \mathcal{B}_{\mathrm{R}}(C_m, \delta_{\mathrm{L}}(C_m), k_3')$ otherwise (i.e. in Steps R.1-R.3). Informally speaking, $\mathcal{B}_2'$ and $\mathcal{B}_3'$ are the branches $\mathcal{B}_2$ and $\mathcal{B}_3$, respectively, restricted to the vertices up to the corresponding cut k_2' or k_3'. The vertex v selected to be in C_1 is called the *starting vertex.*

The branches are used in the subsequent iterations of the algorithm in the way presented in Figure 3, where C_{s_i} refers to the expansion obtained at the end of Step L.i of an iteration of CP, $i = 1, 2, 3$ (the execution of Steps R.1-R.3 is symmetric), and C_{s_0} is the expansion from the beginning of the iteration. First, a branch $\mathcal{B}_1$ is used to obtain C_{s_1} from C_{s_0} (during the execution of Step L.1 of CP). It holds in particular $C_{s_1} = C_{s_0} \cup V(\mathcal{B}_1)$, as stated in Lemma 5 below. It is guaranteed that $r(\delta_{\mathrm{L}}(C_{s_1})) \leq k_1$. The (external) vertices in $V(\mathcal{B}_1) \cap V_{k_1}$ have some neighbors in $V_{k_1+1} \setminus C_{s_1}$ and the algorithm calculates the right branch $\mathcal{B}_2'$ ('emanating' from k_1) in Step L.2. Its right extremity, k_2', may by strictly less than the left extremity of the new right border $\delta_{\mathrm{R}}(C_{s_2})$ if $\mathcal{B}_2'$ has no external vertices in $V_{k_2'}$. Finally, a branch $\mathcal{B}_3'$ is calculated in a symmetric way (this step is omitted if the vertices in $C_{s_1} \cap V_{k_1}$ have no neighbors in $V_{k_1-1} \setminus C_{s_1}$, and in such case $\delta_{\mathrm{L}}(C_{s_3}) \subseteq \delta_{\mathrm{L}}(C_{s_0})$).

The following lemmas are used to prove that the computation stops and they also demonstrate how the expansions change between the subsequent calls of EE.

▶ **Lemma 4.** *Given an expansion C_j and $X \subseteq \delta(C_j)$, after the execution of the ith iteration of the procedure* EE$(r(X), k)$*, where $k \leq r(X)$ (respectively* EE$(l(X), k)$*, where $k \geq l(X)$) it holds $C_{j+i} = C_j \cup V(\mathcal{B}_{\mathrm{L}}(C_j, X, r(X) - i))$ ($C_{j+i} = C_j \cup V(\mathcal{B}_{\mathrm{R}}(C_j, X, l(X) + i))$, respectively), $i \geq 1$.* △

▶ **Lemma 5.** *Let C_{s_0} be an expansion from the beginning of an iteration of* CP*, and let C_{s_i}, $i = 1, 2, 3$, be the expansions obtained at the end of Steps L.1, L.2 and L.3 or R.1, R.2*

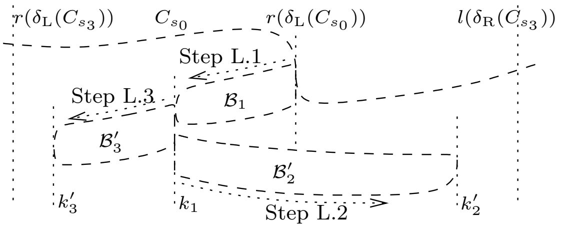

$r(\delta_{\mathrm{L}}(C_{s_3}))$
C_{s_0}
$r(\delta_{\mathrm{L}}(C_{s_0}))$
$l(\delta_{\mathrm{R}}(C_{s_3}))$
Step L.1
Step L.3
$\mathcal{B}_1$
$\mathcal{B}'_3$
$\mathcal{B}'_2$
k'_3
k_1
Step L.2
k'_2

Figure 3 The expansion C_{s_3} obtained from C_{s_0} by including three branches $\mathcal{B}_1, \mathcal{B}'_2$ and $\mathcal{B}'_3$ calculated in Steps L.1,L.2 and L.3 of CP, respectively

and R.3 in this iteration, respectively. Then, $C_{s_1} = C_{s_0} \cup V(\mathcal{B}_1)$, $C_{s_2} = C_{s_1} \cup V(\mathcal{B}'_2)$ *and* $C_{s_3} = C_{s_2} \cup V(\mathcal{B}'_3)$. *Moreover,* $C_{s_3} \neq C_{s_0}$. △

Figure 4 gives an example of the execution of CP. In all cases (including the following figures) ◇ and □ are used to denote the vertices of the right and left borders, respectively. In particular Figure 4(a) presents a graph $\mathcal{G}$ and C_2 (this is the expansion obtained at the end of initialization of CP, where the starting vertex and its neighbor in C_1 are among the three vertices in C_2). Figures 4(b)-(d) depict the state of the algorithm at the end of the first three iterations. (The fourth iteration executes the Steps L.1-L.3, which ends the computation.)

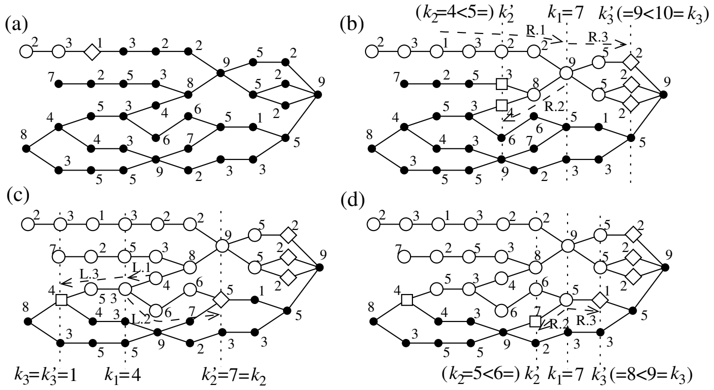


Figure 4 a graph $\mathcal{G}$ (the integers are vertex weights) with distinguished vertices in C_m representing the state of CP after: (a) the initialization; (b) first iteration with Steps R.1-R.3 executed; (c) second iteration with Steps L.1-L.3 executed; (c) third iteration with Steps R.1-R.3 executed

The lemma below follows directly from the instructions in procedure EE.

▶ **Lemma 6.** $\delta(C_j) = \delta_{\mathrm{L}}(C_j) \cup \delta_{\mathrm{R}}(C_j)$ *for each* $j = 1, \ldots, m$. △

The connectedness of $\mathcal{C}$ is due to the fact that $\mathcal{G}[C_j]$ is connected for each $j = 1, \ldots, m$, while the fact that $\mathcal{C}$ is a path decomposition follows from the definition of $\mathcal{G}$.

▶ **Lemma 7.** *Given a simple graph* G *and its path decomposition* $\mathcal{P} = (X_1, \ldots, X_d)$, CP *returns a connected path decomposition* $\mathcal{C} = (Z_1, \ldots, Z_m)$ *of* G. △

4 The approximation guarantee of the algorithm

In this section we analyze the width of the path decomposition $\mathcal{C}$ calculated by CP for the given G and $\mathcal{P}$. First we introduce the concept of a nested expansion, which, informally speaking, is as follows. The first condition for C to be nested states that the weight of $V_i \cap C$ for any i 'between' the right extremity of the left border and the left extremity of the right border (by Lemma 9 the former is less than the latter) is greater than or equal to the weight of the left or the right border of C. The remaining conditions refer to the situation 'inside' borders and are analogous in both cases. The condition (ii) for the left border requires that the weight of $V_i \cap C$, where $i \leq r(\delta_{\mathrm{L}}(C))$, is not less than the weight of the left border restricted to the vertices in $V_1 \cup \cdots \cup V_i$. Finally, condition (iii) gives a 'local' minimality, that is, if we take a left branch $\mathcal{B}_{\mathrm{L}}(C, \delta_{\mathrm{L}}(C), i)$ (where i by the definition is $\leq r(\delta_{\mathrm{L}}(C))$) and we include several vertices of the branch, as it is done in procedure EE, then we 'arrive' at some cut of this branch, and (iii) for C guarantees that the weight of the left border of the new expansion is greater than or equal to the weight of the left border of C.

We say that an expansion C is *nested* if it satisfies the following conditions:

(i) for each $i = r(\delta_{\mathrm{L}}(C)), \ldots, l(\delta_{\mathrm{R}}(C))$, $\min\{\omega(\delta_{\mathrm{L}}(C)), \omega(\delta_{\mathrm{R}}(C))\} \leq \omega(V_i \cap C)$,

(ii) for each $i \leq r(\delta_{\mathrm{L}}(C))$, $\omega(V_i \cap C) \geq \sum_{j \leq i} \omega(V_j \cap \delta_{\mathrm{L}}(C))$, and for each $i \geq l(\delta_{\mathrm{R}}(C))$, $\omega(V_i \cap C) \geq \sum_{j \geq i} \omega(V_j \cap \delta_{\mathrm{R}}(C))$,

(iii) $r(\delta_{\mathrm{L}}(C))$ $(l(\delta_{\mathrm{R}}(C)))$ is a bottleneck of each branch $\mathcal{B}_{\mathrm{L}}(C, \delta_{\mathrm{L}}(C), i)$ (respectively, $\mathcal{B}_{\mathrm{R}}(C, \delta_{\mathrm{R}}(C), i)$).

Figure 5 presents a subgraph of $\mathcal{G}$ on the vertices that belong to an expansion C. For this expansion to be nested it holds in particular: (ii) implies $\omega(V_{i+1} \cap C) \geq \omega(\{x_1, x_2, x_3\})$, $\omega(V_{i+3} \cap C) \geq \omega(\{x_1, x_2, x_3\})$; (i) implies $\omega(V_{i+6} \cap C) \geq \min\{\omega(\delta_{\mathrm{L}}(C)), \omega(\delta_{\mathrm{R}}(C))\} = \min\{\omega(\{x_1, \ldots, x_4\}), \omega(\{y_1, \ldots, y_4\})\}$.

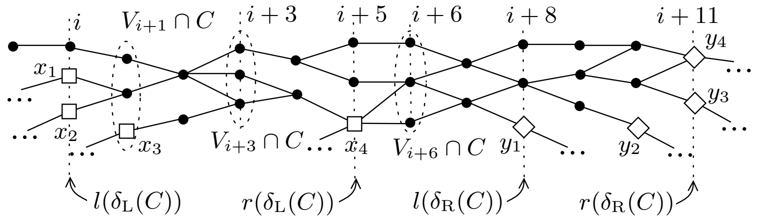


Figure 5 A nested expansion with left and right borders distinguished

Not all expansions computed by CP are nested, but we prove that all of them satisfy (ii) (Lemmas 10-12). The fact that the expansions obtained in Steps I.1-I.3 of CP satisfy (ii) follows from the observation that both the left and right border or each such expansion is a subset of a single set V_i. For this reason we focus on analyzing the subsequent iterations of CP, and we proceed with an assumption that an expansion from the beginning of an iteration (i.e. obtained at the end of the previous iteration, or at the end of Step I.3) is nested. We justify this assumption in Lemma 12.

First we introduce the following concept of moving the borders of an expansion. We say that C_j *moves* the right (left) border of C_{j-1} if $l(\delta_{\mathrm{R}}(C_j)) > l(\delta_{\mathrm{R}}(C_{j-1}))$ $(r(\delta_{\mathrm{L}}(C_j)) < r(\delta_{\mathrm{L}}(C_{j-1}))$, respectively). The following lemma states that in each iteration of CP at most one expansion may be computed that does not move the left or right border of its predecessor. Moreover, this is the first expansion calculated in Step L.2 or in Step R.2, depending on the condition checked in the 'if' statement in the main loop of CP.

▶ **Lemma 8.** *If C_j, $j \in \{2, \ldots, m\}$, is any expansion calculated in Step EL (Step ER) of* EE *invoked in an iteration of* CP*, except for an expansion obtained in the first iteration of* EE *executed in Step L.2 or in Step R.2 of* CP*, then C_j moves the left (right, resp.) border of C_{j-1}.* △

Lemma 6 states that $\delta(C_j) = \delta_{\mathrm{L}}(C_j) \cup \delta_{\mathrm{R}}(C_j)$ for each expansion obtained in CP, while the following lemma implies that the left and right borders of any expansion obtained during the execution of CP are disjoint.

▶ **Lemma 9.** *$r(\delta_{\mathrm{L}}(C_j)) < l(\delta_{\mathrm{R}}(C_j))$ for each $j = 1, \ldots, m$.* △

Provided that an expansion from the beginning of an iteration of CP is nested, the following, together with Lemma 8, implies that if the first expansion computed in Step L.2 or R.2 of CP satisfies (ii), then all expansions from the iteration satisfy (ii).

▶ **Lemma 10.** *Let $j \in \{2, \ldots, m\}$. If C_{j-1} satisfies (ii) and C_j moves the right or the left border of C_{j-1}, then C_j satisfies (ii).* △

Due to the above, we analyze the first expansion obtained in Step L.2 or R.2 of CP. Let C_{s_0} be the expansion from the beginning of an iteration of CP and let C_{s_1} be the expansion obtained at the end of Step L.1 or Step R.1 of this iteration. We obtain that if C_{s_0} is nested, then C_{s_1} satisfies (i) (the proof is omitted due to lack of space). This allows us to prove that the first expansion obtained in Step L.2 or R.2 of CP also satisfies (ii). Thus, due to Lemmas 8 and 10 we obtain that each expansion in an iteration of CP satisfies (ii), when the expansion from the beginning of the iteration is nested.

▶ **Lemma 11.** *If an expansion C_{s_0} from the beginning of an iteration of CP is nested, then each expansion computed by CP in this iteration satisfies (ii).* △

Finally, we finish our argument that each expansion calculated by CP satisfies (ii), by proving the following.

▶ **Lemma 12.** *Let C_{s_0} and C_{s_3} be the expansions from the beginning of two consecutive iterations of CP. If C_{s_0} is nested, then C_{s_3} is nested.* △

Therefore, an induction on the number of iterations of CP allows us to prove the claim that each expansion computed by CP satisfies (ii), as shown in Lemma 14. Lemma 13 below, together with Lemma 6, gives an upper bound for $\omega(\delta(C_j))$ for each $j = 1, \ldots, m$.

▶ **Lemma 13.** *If an expansion C satisfies (ii), then $\omega(\delta_{\mathrm{L}}(C)) \leq \mathrm{width}(\mathcal{G})$ and $\omega(\delta_{\mathrm{R}}(C)) \leq \mathrm{width}(\mathcal{G})$.*

Proof. Suppose w.l.o.g. that $\omega(\delta_{\mathrm{R}}(C)) \geq \omega(\delta_{\mathrm{L}}(C))$. Then, it is enough to argue that $\omega(\delta_{\mathrm{R}}(C)) \leq \mathrm{width}(\mathcal{G})$. To that end observe that by (ii), where $i = l(\delta_{\mathrm{R}}(C))$, $\omega(V_i \cap C) \geq \sum_{k \geq i} \omega(\delta_{\mathrm{R}}(C) \cap V_k) = \omega(\delta_{\mathrm{R}}(C))$. Since $\omega(V_i \cap C) \leq \omega(V_i) \leq \mathrm{width}(\mathcal{G})$, the thesis follows. ◀

▶ **Lemma 14.** *If $\mathcal{C} = (Z_1, \ldots, Z_m)$ is a path decomposition calculated by CP for the given G and $\mathcal{P}$, then $\mathrm{width}(\mathcal{C}) \leq 2 \cdot \mathrm{width}(\mathcal{P}) + 1$.*

Proof. The expansion obtained at the end of Step I.3 of CP is nested. Indeed, (i) and (iii) follow from the fact that a_0' and b_0' are the bottlenecks of the corresponding branches used in Steps I.2 and I.3, respectively, while (ii) trivially holds, for both the left and right border is contained in a single set V_i. Using an induction (on the number of iterations of CP) we obtain by Lemma 12 that any expansion from the beginning of an iteration of CP is nested. Note that for each expansion C_j obtained in Steps I.1-I.3 of CP it holds $\delta_{\mathrm{L}}(C_j) \subseteq V_i$ and $\delta_{\mathrm{R}}(C_j) \subseteq V_{i'}$ for some $i, i' \in \{1, \ldots, d\}$, which implies (ii) for C_j. This, together with Lemma 11, implies that C_j satisfies (ii) for each $j = 1, \ldots, m$. By Lemmas 6 and 9, $\omega(\delta(C_j)) = \omega(\delta_{\mathrm{L}}(C_j)) + \omega(\delta_{\mathrm{R}}(C_j))$ for each $j = 1, \ldots, m$. By Lemma 13, $\omega(\delta(C_j)) \leq 2 \cdot \mathrm{width}(\mathcal{G})$. By the definition, $\mathrm{width}(\mathcal{C}) = \max\{\omega(\delta(C_j)) \colon j = 1, \ldots, m\} - 1$. Thus, by the definition, $\mathrm{width}(\mathcal{C}) \leq 2 \cdot \mathrm{width}(\mathcal{G}) - 1 = 2 \cdot \mathrm{width}(\mathcal{P}) + 1$. ◀

▶ **Lemma 15.** *Let G be a simple connected graph and let $\mathcal{P} = (X_1, \ldots, X_d)$ be its path decomposition of width k. The running time of* `CP` *executed for G and $\mathcal{P}$ is $O(dk^2)$.*

Proof. Since each edge of G is contained in one of the bags of $\mathcal{P}$, $|E(G)| \leq dk$. The number of vertices and edges in $\mathcal{G}$ is $O(kd)$ and $O(dk^2)$, respectively. Thus, the complexity of constructing $\mathcal{G}$ is $O(dk^2)$.

If a branch is given, then the weights of all its cuts can be calculated in time linear in the number of edges and vertices of the branch. The time of finding any branch $\mathcal{B}$ in an iteration of `CP` is $O(|E(\mathcal{B})|)$. The complexity of calculating the weight of all cuts of $\mathcal{B}$, and thus finding its bottleneck, is $O(|E(\mathcal{B})|)$. Whenever two branches overlap, we do not have to repeat the computation. Therefore, the time complexity of determining all branches and their bottlenecks is $O(dk^2)$. This includes the complexity of all executions of the procedure `EE`, because, by Lemma 5, the procedure 'follows' the previously calculated branches by including their vertices into the expansions C_j. It holds that $m \leq kd$, because (by Lemmas 4 and 5) $C_j \subseteq C_{j+1}$ and $C_j \neq C_{j+1}$ for each $j = 1, \ldots, m-1$. By Lemma 14, $\omega(C_j) = O(k)$ for each $j = 1, \ldots, m$. Thus, $\sum_{1 \leq j \leq m} |Z_j| = O(dk^2)$. Thus, the complexity of `CP` is $O(dk^2)$. ◀

▶ **Theorem 16.** *There exists a $O(dk^2)$-time algorithm that for given connected graph G and its path decomposition $\mathcal{P} = (X_1, \ldots, X_d)$ of width k returns a connected path decomposition $\mathcal{C} = (Z_1, \ldots, Z_m)$ such that $\mathrm{width}(\mathcal{C}) \leq 2 \cdot \mathrm{width}(\mathcal{P}) + 1$ and $m \leq kd$.*

Proof. The correctness of `CP` is due to Lemma 7. The inequality $\mathrm{width}(\mathcal{C}) \leq 2\mathrm{width}(\mathcal{P}) + 1$ follows from Lemma 14, and the complexity of `CP` is due to Lemma 15. As argued in the proof of Lemma 15, $m \leq kd$. ◀

▶ **Theorem 17.** *For each connected graph G,* $\mathrm{cpw}(G) \leq 2 \cdot \mathrm{pw}(G) + 1$. ◀

The inequalities $\mathrm{pw}(G) \leq \mathtt{s}(G) \leq \mathrm{pw}(G) + 2$ and $\mathrm{cpw}(G) \leq \mathtt{cs}(G) \leq \mathrm{cpw}(G) + 2$ [4] and Theorem 17 give the following

▶ **Corollary 18.** *For each graph G it holds* $\mathtt{cs}(G) \leq 2 \cdot \mathtt{s}(G) + 3$. ◀

5 Conclusions

The advances in graph theory presented in this paper are three-fold:

- A bound for connected pathwidth is given, $\mathrm{cpw}(G) \leq 2\mathrm{pw}(G) + 1$, where G is any graph, which bounds the connected search number of a graph by its search number, $\mathtt{cs}(G) \leq 2\mathtt{s}(G) + 3$. Moreover, the input vertex v that belongs to the first bag in the resulting connected path decomposition is selected arbitrarily, which implies a stronger fact, namely a connected $(2\mathtt{s}(G) + 3)$-search strategy can be constructed with any vertex of G playing the role of the homebase. This provides an efficient algorithm for converting a search strategy into a connected one with an arbitrary homebase.
- An efficient method is given for calculating a connected pathwidth of width at most $2k + 1$, provided that a graph G and its path decomposition of width k are given.
- It is a strong assumption that the algorithm requires a path decomposition to be given, because calculating $\mathrm{pw}(G)$ is a hard problem in general. However, this algorithm can be used to approximate the connected pathwidth for the classes of graphs for which the approximate algorithms for pathwidth exist.

References

1 L. Barrière, P. Flocchini, P. Fraigniaud, and N. Santoro. Capture of an intruder by mobile agents. In *SPAA '02: Proceedings of the fourteenth annual ACM symposium on parallel algorithms and architectures*, pages 200–209, New York, NY, USA, 2002. ACM.

2 L. Barrière, P. Fraigniaud, N. Santoro, and D.M. Thilikos. Connected and internal graph searching. Technical report, Technical Report, UPC Barcelona, 2002.

3 L. Barrière, P. Fraigniaud, N. Santoro, and D.M. Thilikos. Searching is not jumping. In *WG '03: Proceedings of the 29th International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 34–45, 2003.

4 D. Bienstock. Graph searching, path-width, tree-width and related problems (a survey). *DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, 5:33–49, 1991.

5 H.L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–22, 1993.

6 D. Dereniowski. Connected searching of weighted trees. In *MFCS*, 2010. (to appear).

7 Fedor V. Fomin, Pierre Fraigniaud, and Dimitrios M. Thilikos. The price of connectedness in expansions. Technical report, Technical Report, UPC Barcelona, 2004.

8 F.V. Fomin and D.M. Thilikos. An annotated bibliography on guaranteed graph searching. *Theor. Comput. Sci.*, 399(3):236–245, 2008.

9 P. Fraigniaud and N. Nisse. Connected treewidth and connected graph searching. In *Proc. of the 7th Latin American Symposium on Theoretical Informatics (LATIN'06), LNCS*, volume 3887, pages 479–490, Valdivia, Chile, 2006.

10 P. Fraigniaud and N. Nisse. Monotony properties of connected visible graph searching. *Inf. Comput.*, 206(12):1383–1393, 2008.

11 J. Gustedt. On the pathwidth of chordal graphs. *Discrete Appl. Math.*, 45(3):233–248, 1993.

12 T. Kashiwabara and T. Fujisawa. Np-completeness of the problem of finding a minimum-clique-number interval graph containing a given graph as a subgraph. In *Proc. IEEE Inter. Symp. Circuits and Systems*, pages 657–660, 1979.

13 N.G. Kinnersley. The vertex separation number of a graph equals its path-width. *Inf. Process. Lett.*, 42(6):345–350, 1992.

14 L.M. Kirousis and C.H. Papadimitriou. Interval graphs and searching. *Discrete App. Math.*, 55:181–184, 1985.

15 L.M. Kirousis and C.H. Papadimitriou. Searching and pebbling. *Theor. Comput. Sci.*, 47(2):205–218, 1986.

16 N. Megiddo, S.L. Hakimi, M.R. Garey, D.S. Johnson, and C.H. Papadimitriou. The complexity of searching a graph. *J. ACM*, 35(1):18–44, 1988.

17 R. Mihai and I. Todinca. Pathwidth is NP-hard for weighted trees. In *FAW '09: Proceedings of the 3d International Workshop on Frontiers in Algorithmics*, pages 181–195, Berlin, Heidelberg, 2009. Springer-Verlag.

18 R. Möhring. Graph problems related to gate matrix layout and PLA folding. In *E. Mayr, H. Noltemeier, and M. Syslo eds, Computational Graph Theory, Computing Supplementum*, volume 7, pages 17–51, 1990.

19 N. Nisse. Connected graph searching in chordal graphs. *Discrete Applied Math.*, 157(12):2603–2610, 2008.

20 S.L. Peng, M.T. Ko, C.W. Ho, T.S. Hsu, and C.Y. Tang. Graph searching on chordal graphs. In *ISAAC '96: Proceedings of the 7th International Symposium on Algorithms and Computation*, pages 156–165, London, UK, 1996. Springer-Verlag.

21 N. Robertson and P.D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

22 B. Yang, D. Dyer, and B. Alspach. Sweeping graphs with large clique number. *Discrete Mathematics*, 309(18):5770–5780, 2009.

Polynomial Fitting of Data Streams with Applications to Codeword Testing*†

Andrew McGregor[a], **Atri Rudra**[b], **and Steve Uurtamo**[c]

a Department of Computer Science,
University of Massachusetts, Amherst, MA.
mcgregor@cs.umass.edu

b,c Department of Computer Science and Engineering,
University at Buffalo, SUNY, Buffalo, NY.
{atri,uurtamo}@buffalo.edu

Abstract

Given a stream of (x, y) points, we consider the problem of finding univariate polynomials that best fit the data. Over finite fields, this problem encompasses the well-studied problem of decoding Reed-Solomon codes while over the reals it corresponds to the well-studied polynomial regression problem.

We present one-pass algorithms for two natural problems: i) find the polynomial of a given degree k that minimizes the error and ii) find the polynomial of smallest degree that interpolates through the points with at most a given error bound. We consider a range of error models including the average error per point, the maximum error, and the number of points that are not fitted exactly. Many of our results apply to both the reals and finite fields. As a consequence we also solve an open question regarding the tolerant testing of codes in the data stream model.

Keywords and phrases Streaming, Polynomial Interpolation, Polynomial Regression

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.428

1 Introduction

In this paper we consider the following problem: given a stream of n input points $(x_i, y_i) \in \mathbb{F}^2$ (where all the x_i's are distinct[1]) for some field $\mathbb{F}$, fit them with a univariate polynomial with low error. This general problem has been intensively studied under at least two broad specifications. The first case is when $\mathbb{F}$ is the set of reals and we want to minimize the least squares or least absolute deviation errors – in this setting the problem is called (polynomial) regression. The second case is when $\mathbb{F}$ is a finite field and we are trying to minimize the number of disagreements– this corresponds to the problem of decoding Reed-Solomon codes. Both of these problems have great practical value: regression is used to build a succinct model of the input points and is perhaps the most widely used statistical tool. Reed-Solomon codes are widely used to guard against corruption of data in everyday use such in communication protocols and in storage media. We present data stream algorithms for both problems.

The case for data stream algorithms as a tool to handle massive data sets has been well made over the last couple of decades (see, e.g., the survey by Muthukrishnan [14]).

* A. McGregor's research was supported by NSF CAREER Award CCF-0953754, while A. Rudra and S. Uurtamo were supported by NSF CAREER Award CCF-0844796.

† The first and the second authors thank the NSF for inviting them to a meeting where the authors first started discussing the problems considered in this paper.

[1] Some of our results hold even without this condition but for simplicity, we make this assumption. We also note that checking whether this assumption holds or not requires $\Omega(n)$ space.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 428–439
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

As polynomial fitting is an extremely basic problem, it is natural to consider its data stream complexity. In addition, these problems also have practical motivations. Polynomial regression dates back to work of Legendre and Gauss and can be used to make sense of large data sets or to use a large experimental data set to accurate estimate model parameters, see, e.g., work in epidemiology [7] and marine geography [1]. Polynomial regression is actually a special case of multivariate linear regression since some variables may be polynomial functions of other variables. Multivariate linear regression has been recently considered in the data stream model but existing work is either only applicable to least squares polynomial regression [3] (and we'll observe that a simpler approach works in this case) or is not as time-efficient [6]. By focusing on a special case of the regression problem we are able to a) consider a wider range of error measures such as maximum absolute error and cardinality of errors, b) develop faster and more space-efficient algorithms, c) consider variants of the problem such as fitting the simplest model subject to an error budget, and d) consider fields other than the reals.

Approximation of decoding codes under the umbrella of property testing has been intensively studied since the the advent of interactive proofs more than two decades ago. While Reed-Solomon codes are less interesting from a sampling perspective, the second and third authors recently introduced the problem of designing data stream algorithms for codeword testing [15]. They detailed applications of data stream algorithms for codeword testing in storage systems and network systems. We discuss this further in Section 1.1.

The problems of polynomial regression and decoding of Reed-Solomon codes have inherently different motivations. In particular, let k be the degree of the polynomial that we are trying to fit through the data. In polynomial regression we want to make k as small as possible as that means that our data has a small representation. On the other hand, for Reed-Solomon codes we want k to be as large as possible as that means we introduce as little redundancy as possible. Further, different kinds of error make sense in the two problems. For the Reed-Solomon case, which are defined over finite fields, the measure of error is the Hamming distance, or the ℓ_0 norm. On the other hand, for polynomial regression, ℓ_p and ℓ_∞ measures also make sense. These differences crop up in the kind of algorithms generally used to tackle these two problem. Many of our solutions, however, are "oblivious" as to whether we are working over finite fields or the reals. Some proofs were omitted for space considerations, however, all omitted proofs are available in the full version of the paper.

1.1 Decoding of Reed-Solomon Codes and Related Problems

We now focus on the polynomial fitting problem when $\mathbb{F}$ is a finite field. In this case, we will primarily consider the error distance of Hamming distance (i.e. the number of positions where the fitted polynomial disagrees with in the input point). However, some of our results also extend to the ℓ_p case for $p > 0$.

First consider the problem of *error detection*, i.e., we want to figure out if a polynomial of degree at most k fits exactly through all the n input points. Even though much weaker than the error correction problem, error detection is widely used in practice, e.g., in Internet traffic where one uses a checksum to detect errors. In fact, the error detection algorithm (compute the checksum of the data and compare it with the stored checksum) is a very efficient one-pass data stream algorithm. This feature of checksums is hugely attractive in practice even though checksums have terrible error-correction properties. One of the motivations of [15] was to see if the Reed-Solomon code, which has excellent error-correction capabilities, could also have data stream algorithms for error detection. It was shown in [15] that this problem does indeed have a poly-log space, single-pass data stream algorithm if we allow the algorithm advance knowledge of the x_i's.

Given the somewhat surprising fact that error detection for Reed-Solomon codes does indeed have efficient data stream algorithms, [15] also looked at the following *tolerant testing* problem: Is it the case that there exists a polynomial of degree at most k that disagrees with at most t points in the input or do all polynomials of degree at most k have to disagree in at least $2t$ points? They showed that the trivial algorithm of trying out all possible error locations can be implemented in $\tilde{O}(t)$ space with one pass. They also used group testing ideas to improve the running time with similar parameters under the additional constraint of $tk \leq O(n)$. It was an open question whether this could be improved (and, in fact, the second author has widely conjectured that one would need $\tilde{\Omega}(t)$ space). In this paper, we show that one can in fact solve this problem in $\tilde{O}(k)$ space *independent* of t.

The main building block for the algorithm above is an algorithm to estimate the F_0 value of a vector where each coordinate is updated by an addition over the field $\mathbb{F}$. This problem of course, is very well studied for the case of the reals and has been implicitly studied for finite fields [10]. Our algorithm is similar to existing algorithms for estimating F_0 (see, e.g., [12]) and works for any field $\mathbb{F}$. The technical ingredient is a subroutine to determine efficiently if a given subset of vector positions has a nonzero value in it (the catch is that for fields in general nonzero elements can add up to zero). Given this subroutine, the algorithm mentioned in the paragraph above is simple: sketch the input y values and then cycle through all the q^k possibilities for the codewords. (The latter can be done in low space if the algorithm has full knowledge of the x_i's.) In general, trying to improve the running time of this algorithm is hopeless as it would solve the maximum likelihood problem for Reed-Solomon codes, i.e., computing the degree k polynomial that disagrees with the minimum number of input points), which is known to be NP-hard [8]. In fact, there are no known approximate maximum likelihood algorithms for any nontrivial codes, and this is a notoriously hard problem [5]. However, under additional constraints on t, we show that one can in a single pass, compute the closest polynomial and estimate t in space $\tilde{O}(k)$.

We also consider the following natural problem related to polynomials: given the coefficients of a degree k polynomial, compute the number of roots of the polynomial. Note that in this case the trivial algorithm of storing the entire input takes $\tilde{O}(k)$ space. In fact, computing the number of non-roots is the same as computing the F_0 value of the stream of the evaluation of the polynomial over all elements of the field, which by our earlier algorithm is easy. However, we show that the complementary problem of computing the number of roots takes $\Omega(k)$ space. In fact, this is true even if we want to solve the simple problem of checking if the polynomial has any roots at all. The reduction is from set-disjointness and makes use of some properties of non-squares in fields.

1.2 Polynomial Regression

We now discuss our results for polynomial regression over the reals (though some of our results work over any field). Given $p \geq 0$, and n points $(x_1, y_1), \ldots, (x_n, y_n)$, the aim is to find a univariate polynomial $f(X)$ of degree at most k that minimizes the ℓ_p error, i.e. the sum $\sum_{i \in [n]} |y_i - f(x_i)|^p$. A fairly easy argument shows that one requires $\Omega(k)$ space to solve this problem. We show that under many scenarios, this lower bound is indeed tight. Contrast this with the general regression problem where the corresponding space bound is $\tilde{\Theta}(k^2)$ [3].

We first consider the case when we are given a bound e on the error we are willing to tolerate and we are interested in computing $f(X)$ of the smallest possible degree k that results in an error of at most e. Note that in this case k is *unknown* but we still want to use space that is $\tilde{O}(k)$. We present two one pass $\tilde{O}(k+e)$-space algorithms for the case when

$e = 0$ and $e > 0$ but $p = 0$. In both cases, our algorithms work under extra conditions on n, k and e. The main idea in both of these algorithms is to build an estimate $\hat{k}$ on k progressively. It is not too hard to figure out when the estimate $\hat{k}$ is smaller than the actual value of k. Our main insight is that under suitable conditions on n, k and e, when we discover that our estimate $\hat{k}$ is insufficient, we can discard all we have seen so far and start from scratch. The conditions guarantee that we never throw away too much information. Also crucial to our algorithms are the known observations that (i) Newton's interpolation formula can be implemented in an "online" fashion and uses $\tilde{O}(k)$ space and (ii) existing decoding algorithms for Reed-Solomon codes can be implemented in linear space.

Next we consider the case when k is specified up-front. For $p = 0$, our algorithm samples $O(k)$ points and then uses the decoding algorithm for Reed-Solomon codes mentioned in the previous paragraph to compute the optimal polynomial. For $p \in [1, 2)$, we use Indyk's p-stable sketching technique to sketch the y and x values. A naive approach would then be to cycle through all possible values for the coefficients (we also provide a simple one pass algorithm to bound the range of values each coefficient can take). While this results in an $\tilde{O}(k)$ space algorithm, the running time is not satisfactory. Using the convexity of the error function, we present an algorithm that effectively does a binary search in a k-dimensional space to compute the best values of the coefficients. This leads to a $O(\log^k n)$-pass $\tilde{O}(k)$ space algorithm, which we then refine to a one-pass, $\tilde{O}(k)$ space and $O(\log^k n)$ time algorithm. Finally, we consider the case of $p = \infty$. We observe that a result due to Chan and Chen [2] can be used to solve the problem *exactly* in constant passes and sub-linear space. We also present a one-pass $\tilde{O}(k)$ space algorithm to approximate the ℓ_∞ error that is in turn based on the fact that one can compute the optimal polynomial for any *even* $p \geq 2$ with one pass and $\tilde{O}(p^2 k)$ space.

2 Finding Smallest Degree Polynomials

We first consider the following problem: Given n input points (x_i, y_i) $(1 \leq i \leq n)$ and an integer $0 \leq e \leq n$, compute the polynomial $f(X)$ of the smallest degree k such that $|\{i | f(x_i) \neq y_i\}| \leq e$.

2.1 Perfect interpolation

We begin with a one-pass $\tilde{O}(k)$ space algorithm to compute the polynomial of minimum degree k that interpolates through all the points, i.e., we solve the problem above with $e = 0$. This will serve as a warmup for the general case (in addition to giving a slightly better result for this special case.) Note that here we do *not* know k in advance and this is what makes the problem nontrivial. Furthermore, we do not make any assumptions about the range or order of the points nor do we know $\{x_i : i \in [n]\}$ in advance.

▶ **Theorem 1.** *Let $(x_1, y_1), \ldots, (x_n, y_n)$ be n input points such that there is an unknown polynomial $f(X)$ such that for every $1 \leq i \leq n$, $f(x_i) = y_i$. Then there exists a one-pass $\tilde{O}\left(\frac{1}{\epsilon} \cdot \deg(f)\right)$ space algorithm to compute $f(X)$, provided* $\deg(f) \leq (1/2 - \epsilon)n$. *The amortized update time of the algorithm is* $O(\deg(f))$.

We will use the following well-known result crucially in our algorithm:

▶ **Proposition 2.** *Let the points $(x_1, y_1), \cdots, (x_m, y_m)$ be explained by a polynomial $P(X)$ of degree at most m. Then the points $(x_1, y_1), \ldots, (x_m, y_m), (x_{m+1}, y_{m+1})$ are explained by* *the unique polynomial*

$$Q(X) = P(X) + (y_{m+1} - P(x_{m+1})) \cdot \prod_{i=1}^{m} \frac{X - x_i}{x_{m+1} - x_i}.$$

Further, $\deg(Q) \in \{\deg(P), m\}$.

It is easy to verify that the polynomial $Q(X)$ does indeed work – its uniqueness follows from the fact that two distinct polynomials of degree at most m can agree in at most m points. Further, the claim on the degree of $Q(X)$ follows from the fact that $Q(X) = P(X)$ if $y_{m+1} = P(x_{m+1})$. Finally, note the following corollary that we will use later on: $Q(X)$ can be computed from just the knowledge of $P(X)$, y_{m+1} and $x_1, \ldots, x_{m+1}$.

Proposition 2 implies the following $O(\log(\deg(f)))$-pass algorithm: guess the degree of $f(X)$ in a geometric series and then use Proposition 2 to fit the data with a polynomial with the guessed degree. Our algorithm achieves the result in a single pass.

Proof of Theorem 1. For notational simplicity define $k = \deg(f)$. The algorithm maintains an estimate k' of k. The algorithm also maintains a polynomial $P(X)$ of degree at most k' that explains the last few points (the exact number will be specified later). Now consider the case when the algorithm sees a new point (x_i, y_i). Two things can happen: (i) $P(x_i) = y_i$. In this case, we are good as the current polynomial $P(X)$ explains the new point; or (ii) $P(x_i) \neq y_i$. In this case we want to use Proposition 2 to compute the new polynomial $Q(X)$. Note that in this second case, $\deg(Q) = m$. However, to compute $Q(X)$, we also need to remember all the x_i's we have seen so far.

To implement the idea for part (ii), we will need to keep track of all the x_i's we have seen so far. However, we cannot store all the x_i values if case (ii) never happens (as in that case we would have stored $\omega(\deg(f))$ values). The main observation is to keep track of $O(k')$ x_i values and in case those are not sufficient enough to compute the new $Q(X)$, we update k' accordingly and restart the whole process. The bound of $k \leq (1/2 - \epsilon)n$ is to make sure that by the time we attain $k' = k$, we still have $k + 1$ points left to interpolate through.

We now present the details of the algorithm. Let $c = O(1/\epsilon)$ be a constant that we will fix later on.

1. Initialize $k' \leftarrow 1$ and let $P(X)$ be the line that passes through (x_1, y_1) and (x_2, y_2).
2. Set $i, j \leftarrow 2$ and $S \leftarrow \{x_1, x_2\}$. The role of i is to count the total number of points seen so far while j counts the number of points seen since last restart.
3. Repeat until $i \leq n - k' - 1$:
   - **a.** Set $i \leftarrow i + 1$ and read (x_i, y_i).
   - **b.** If $P(x_i) == y_i$ then add x_i to S unless $|S| == ck'$. Set $j \leftarrow j + 1$.
   - **c.** Else if $j == |S|$ then set $k' \leftarrow j$ and set $P(X)$ as the $Q(X)$ given by Proposition 2. (Note that this be computed from the existing $P(X)$ and S.) Finally, add x_i to S.
   - **d.** Else set $k' \leftarrow j$ and $j \leftarrow 0$. (the “Restart")
     - Read the points $(x_{i+1}, y_{i+1}), \ldots, (x_{i+k'}, y_{i+k'})$.
     - Set $S \leftarrow \{x_i, \ldots, x_{i+k'}\}$.
     - Set $P(X)$ to be the unique degree at most k' polynomial through $(x_i, y_i), \ldots, (x_{i+k'}, y_{i+k'})$.
     - Set $i \leftarrow i + k'$.
4. If $P(X)$ explains the remaining points then output $P(X)$,
5. Else output $k > (1/2 - \epsilon)n$.

Note that if the algorithm halts and outputs $P(X)$ *and* $k' = k$, then it indeed outputs the correct $f(X)$. This is because $P(X)$ explains at least $k + 1$ points and there is a unique polynomial of degree at most k that explains any $k+1$ points. Further, note that the algorithm can be implemented in one pass and uses space $O(ck)$ assuming $k' = k$ at the end of the algorithm.

To complete the proof, we will show that assuming $k \le (1/2 - \epsilon)n$, the algorithm indeed outputs $f(X)$. Toward this end, we first note that in the algorithm whenever we update k', there exists a polynomial of degree k' that agrees with the last $k'+1$ points. Now, if at any update we get $k' > k$, then it means that two polynomials – one of degree k' (the polynomial $P(X)$) and another of degree $k < k'$ (the polynomial $f(X)$) agree on $k'+1$ points, which is not possible. Thus, we have that at any stage of the algorithm, $k' \le k$. To prove that at the end, $k' = k$, we claim that the last restart happens at $i \le n-k-1$. Assuming this claim is true, note that the algorithm outputs a polynomial $P(X)$ of degree $k' \le k$ that agrees with the last $k+1$ points. This implies that $f(X) = P(X)$ (and hence $k' = k$).

To complete the proof we need to prove that the last restart happens at $i \le n-k-1$. To this end, we will show that the number of items discarded during restarts is at most $n-k-1$. Indeed, we consider the set of indices $\{i_1, \dots, i_m\} \subseteq [n]$, where during the i_ℓth iteration ($\ell \in [m]$), the value of k' changed. For ease of notation, let the k' value at the i_ℓth iteration for $\ell \in [m]$ be denoted by $k'(\ell)$. Note that for every $1 \le \ell < m$, $k'(i_\ell) \le k'(i_{\ell+1})$. Further, call $j \in [m]$ *bad* if k' changed as a result of a restart. Further, note that the number of discarded points is exactly $\sum_{\ell \text{ bad}} k'(\ell)$. Now, note that when ℓ is bad then since we did not go through Step 3(c), we have the current value of j in Step 3(d) satisfying $j > ck'(\ell-1)$. Thus, this implies that for bad ℓ, $k'(\ell) > ck'(\ell-1)$. Further recall that we had shown earlier that $k'(m) \le k$. Thus, the sum above is bounded by $\sum_{i=0} k/c^i = \frac{c}{c-1} \cdot k \le (1+\epsilon)k - 1$, where the last inequality follows by choosing an appropriate $c \in O(1/\epsilon)$. Thus, we would have proved the claim if $(1+\epsilon)k - 1 \le n-k-1$, which in turn is implied by the assumption that $k \le (1/2-\epsilon)n$. ◀

2.2 Interpolation with outliers

We now present a one-pass $\tilde{O}(k)$ space algorithm to compute the polynomial of minimum degree k that interpolates through all but e points.

▶ **Theorem 3.** *Let $(x_1, y_1), \dots, (x_n, y_n) \in \mathbb{F}^2$ be n input points such that there is an unknown polynomial $f(X)$ such that $|\{i | f(x_i) \neq y_i\}| \le e$ for some $0 \le e \le n$. Then there exists a one-pass $\tilde{O}(e + \deg(f))$ space algorithm to compute $f(X)$, provided $(e + \deg(f)) \cdot \log(\deg(f)) \le O(n)$. The amortized update time of the algorithm is $\tilde{O}(k+e)$.*

To prove the theorem above, we will need the error-version of Proposition 2, i.e. a decoding algorithm for Reed-Solomon codes. It is known, for example, that the Berlekamp-Massey algorithm implies the following:

▶ **Theorem 4.** *Let $1 \le K \le N$ be integers. Let $(x_1, y_1), \dots, (x_N, y_N) \in \mathbb{F}^2$ be points. There exists an $\tilde{O}(N)$ space algorithm using $\tilde{O}(N^2)$ field operations that outputs the unique polynomial $P(X)$ of degree at most K such that $|\{i | f(x_i) \neq y_i)\}| < (N-K)/2$.*

The proof of Theorem 3 follows that of Theorem 1, where we use Theorem 4 instead of Proposition 2. The proof is a bit simpler because of the stricter bounds on $\deg(f)$.

3 Polynomial Fitting

In this section we consider finding the degree k polynomial $f(x) = \sum_{i=0}^{k} a_i x^i$ that best fits a stream $(x_1, y_1), \dots, (x_n, y_n)$ of points. We will consider various measures of fit including the total number of points that are not interpolated exactly, the average error on each point, and the maximum error over all points. We introduce the following family of functions $\mathcal{E}_p : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^{k+1} \to \mathbb{R}$

$$\mathcal{E}_p(x, y, a) := \sum_{i=1}^{n} |y_i - f_a(x_i)|^p \quad \text{where} \quad f_a(x) = \sum_{i=0}^{k} a_i x^i$$

and write $\mathcal{E}_p^{(k)}(x, y) := \min_{a_k,\ldots,a_0} \mathcal{E}_p(x, y, a)$. We also write $\mathcal{E}_\infty(x, y, a) := \max_{i\in[n]} |y_i - f_a(x_i)|$ and $\mathcal{E}_\infty^{(k)}(x, y) := \min_{a_k,\ldots,a_0} \mathcal{E}_\infty(x, y, a)$.

We start by noting that the case of minimizing $\mathcal{E}_2$ is actually easy in $O(k \log n)$ bits of space! This is because the optimal choices of the a_i coefficients are determined by the following $k + 1$ equations:

$$a_0 \sum_{i\in[n]} x_i^j + a_1 \sum_{i\in[n]} x_i^{j+1} + \ldots + a_k \sum_{i\in[n]} x_i^{j+k} = \sum_{i\in[n]} x_i^j y_i \qquad \forall\ 0 \le j \le k\ .$$

The equations correspond to the derivatives of $\mathcal{E}_2(x, y, a)$ with respect to each a_j. It is therefore sufficient to compute the following $O(k)$ values

$$\sum_{i\in[n]} x_i^j \quad \text{for} \quad 0 \le j \le 2k\ , \qquad \text{and} \ \sum_{i\in[n]} x_i^j y_i \quad \text{for} \quad 0 \le j \le k$$

as the stream is processed. The resulting set of simultaneous equations are then solved in post processing.

A similar idea works for $p \in \{4, 6, 8, \ldots\}$ but requires a bit more space. The main idea is simple (and has been observed for the more general regression problem): if one thinks of the coefficients $a_0, \ldots, a_k$ as variables then $\mathcal{E}_p(x, y, a)$ is a $(k + 1)$-variate polynomial of degree p. Thus, if we can keep track of all the coefficients in this polynomial, then after the pass over the input, one can estimate $\mathcal{E}_p^{(k)}(x, y)$ by cycling through all possibilities for a. This requires keeping track of roughly p^k values, which is not satisfactory. However, it is easy to check that these roughly p^k coefficients only depend on the following $O(p^2k)$ sums:

$$\sum_{i\in[n]} y_i^j x_i^\ell \qquad \text{for} \qquad 0 \le j \le p, \qquad \text{and} \qquad 0 \le \ell \le (p - j)k.$$

Thus, we only need to keep track of the above $O(p^2k)$ sums and $\mathcal{E}_p^{(k)}(x, y)$ can be evaluated in post-processing.

However, it is not possible to find the best coefficients in general in sublinear space. An easy way to see this is to consider $p = 1$ and $k = 0$. Given a set of points $\{(x_i, y_i) : i \in [n]\}$ we seek the value a such that $\sum_{i\in[n]} |x_i - a|$ is minimized. But it is well known that the optimal value of a is the median of the y_i values and computing the median exactly in the data stream model requires $\Omega(n)$ bits of space [9].

Another simple lower bound shows that if, rather than reporting the best $k+1$ coefficients, we just want to multiplicatively estimate $\mathcal{E}_p^{(k)}(x, y)$ then this requires $\Omega(k)$ bits of space. This follows from a reduction from indexing where Alice has a set $A \in [2k]$ of cardinality k and Bob has an index $j \in [2k]$. Alice computes the degree k polynomial $f(x) = \prod_{a\in A}(x - a)$ and defines the first $k + 1$ elements of a stream $\{(2k + i, f(2k + i)) : i \in [k + 1]\}$. Bob then adds the point $(j, 0)$. If $j \in A$ then there is a degree k polynomial, namely f, that interpolates through all the $k + 2$ points exactly. Alternatively if $j \notin A$ then any interpolating polynomial must have degree at least $k + 1$.

3.1 Maximizing the number of points fitted exactly

We first consider fitting a degree k polynomial to the stream of points with the goal of interpolating exactly through as many of the points as possible. The following result applies to both finite fields $\mathbb{F}_q$ and the reals. The application to finite fields relies on the observation that many sketching algorithms for estimating the number of distinct items, F_0, can be carefully modified to estimate $|\{i : f_i \bmod p \neq 0\}|$ for an arbitrary prime p where f_i is the frequency of the value i in the stream.[2]

▶ **Theorem 5.** *Let $n \geq (1+2\gamma)k$ be integers for $\gamma > 0$. Assume that $\mathcal{E}_0(x, y) \leq ((1-\gamma)n-k)/2$ for some $\gamma > 0$. Then it is possible to find the optimal polynomial and estimate $\mathcal{E}_0(x, y)$ up to a factor $(1 + \epsilon)$ with probability $1 - \delta$ (where $\delta \leq \exp(-\Omega(\gamma^2 k))$) in a single pass using $\tilde{O}(\epsilon^{-2} \log(1/\delta) + \gamma^{-2}k)$ space.*

Proof. Let f be a degree k polynomial that interpolates through the maximum number of points. Note that by the bound on $t = \mathcal{E}_0(x, y)$, f will be unique. Call a point (x_i, y_i) *good* if $f(x_i) = y_i$ and *bad* otherwise. The idea in the algorithm is to essentially sample enough points and run the unique decoding algorithm from Theorem 4 on the sampled points. We next present the details.

First assume that $t \leq \gamma k$. In this case, we just run the algorithm from Theorem 4 with $K = k$ and $N = (1 + 2\gamma)k$ on the first N points. Thus, even if in the worst case all the t errors occur in the first N positions, the algorithm from Theorem 4 will output f in space $\tilde{O}(k)$. Note that once we have computed f, we can check that $t \leq \gamma k$ by verifying that f explains the remaining points. Further, we can compute t exactly.

Next we consider the case when $t > \gamma k$. In this case we first sample each of the n input points with probability $4k/(\gamma^2 n)$. By Chernoff, except with probability $\exp(-\Omega(k))$, we would have sampled $N = ck$ points with $4/\gamma^2(1 - \gamma/2) < c < 5/\gamma^2$. Then we run the algorithm from Theorem 4 on the sampled points. Note that if we sample at most $(ck - k)/2$ bad points, the algorithm will indeed return f. Next, we show that this is indeed the case. Note that the expected number of bad points is $\mu = 4kt/(n\gamma^2)$. We show by a case analysis that the probability we get more than $\Delta := (c - 1)k/2$ bad points is exponentially small.

We first consider the sub-case that $\gamma k \leq t < n/(8e)$. Note that in this case $\Delta/\mu > 2e$, which implies that the probability that the number of bad points is more than Δ is at most $2^{-t} = \exp(-\Omega(k))$ [4]. Finally we consider the sub-case that $t \geq n/(8e)$. Note that by the assumption on t, we also have $t \leq (1 - \gamma)n/2$, which implies that in this case $\mu \leq 4k(1 - \gamma)/(2\gamma^2)$. This implies that $\Delta > (1 + \gamma/2)4k(1 - \gamma)/(2\gamma^2) \geq (1 + \gamma/2)\mu$ (as $c > (1 - \gamma/2)4/\gamma^2 \geq 4/\gamma^2(1 - \gamma/2 - \gamma^2/2) + 1)$. Thus, the probability that we will have at least Δ bad points by the "usual" Chernoff bound is upper bounded by $\exp(-\Omega(\gamma^2 \cdot \mu)) \leq \exp(-\Omega(\gamma^2 n))$. Thus, in a single pass and $\tilde{O}(k/\gamma^2)$ space we can compute the optimal polynomial $f(X) = \sum_{i=0}^{k} a_i X^i$ with error probability at most $\exp(-\Omega(\gamma^2 k))$.

[2] In particular, the algorithm detailed in [12] computes $\sum_{i \in S} f_i$ for random subsets S and makes an estimation based on the fraction of random subsets S such that $\sum_{i \in S} f_i \neq 0$ as this indicates that there exists $i \in S$ such that $f_i \neq 0$. However, in the case of $\mathbb{F}_p$ for example, it is possible that $\sum_{i \in S} f_i = 0 \bmod p$ while there exists $i \in S$ such that $f_i \neq 0 \bmod p$. One approach, as taken in Indyk [10] for the case $p = 2$, is to take the probability of this event into account and adjust the estimator appropriately. An alternative approach is to consider $\log(1/\gamma)$ random subsets of each S, $\{S_j : j \in \log(1/\gamma)\}$: if there exists $i \in S$ such that $f_i \neq 0 \bmod p$ then with probability at least $1 - \gamma$, there exists S_j such that $\sum_{i \in S_j} f_i \neq 0 \bmod p$. The results in a factor $\log(1/\gamma)$ increase in the space and time use of the algorithm but it suffices for γ to be $O(\epsilon^{-2})$ so this increase is not significant.

In parallel with the algorithm above, we compute sketches to estimate the value of t. Compute F_0 sketches (e.g., [13]) of $y = (y_1, \ldots, y_n)$ and $x^j = (x_1^j, x_2^j, \ldots, x_n^j)$ for $0 \leq j \leq k$ and return an estimate based for t using $\text{sketch}(y - \sum_{j=0}^{k} a_j x^j) = \text{sketch}(y) - \sum_{j=0}^{k} a_j \cdot \text{sketch}(x^j)$.

Repeating the above process $O(\log 1/\delta)$ times and taking the smallest estimate gives a $(1 \pm \epsilon)$ approximation on t with error probability at most δ. Note that we use $\tilde{O}(\epsilon^{-2} \log(1/\delta))$ space in this part of the algorithm. The assumption on δ in the statement of the theorem completes the proof. ◀

3.2 Minimizing the average error

To minimize $\mathcal{E}_p(x, y, a)$ for $p \in (0, 2)$, we first consider the case where we may assume that each a_i comes from some set of t discrete values. Using the p-stable sketching technique [11], construct linear sketches of the $k+1$ vectors $x^j = (x_1^j, x_2^j, \ldots, x_n^j)$ for $0 \leq j \leq k$ and $y = (y_1, \ldots, y_n)$. Call these sketches $\text{sketch}(x^j)$ Then for a given setting of $a_0, \ldots, a_k$, we can estimate $\mathcal{E}_p(x, y, a)$ up to factor $1 + \epsilon$ because the sketches are linear:

$$\text{sketch}(y - \sum_{j=0}^{k} a_j x^j) = \text{sketch}(y) - \sum_{j=0}^{k} a_j \cdot \text{sketch}(x^j) \ .$$

If the sketches are of size $\tilde{O}(\epsilon^{-2} \log \delta^{-1})$ then this procedure fails with probability at most δ. Since there are at most t^k settings for a, this procedure works for testing all settings of a with probability at least $1 - t^k \delta$. Rescaling δ gives a $\tilde{O}(\epsilon^{-2} k \log t \log \delta^{-1})$ space algorithm. A similar idea was used in Feldman et al. [6] for multivariate linear regression. The main drawback with this approach is the $O(t^k)$ time required for post-processing.

To ameliorate the situation slightly, we first argue that if we restrict ourselves to finding coefficients up to polynomial precision, we may assume that t is polynomial. In particular we know how to compute a value B in one pass over the input such that all the coefficients of the polynomial $f(X) = \sum_{i=0}^{k} a_i X^i$ minimizing $\mathcal{E}_p(x, y, a)$ satisfy $|a_i| \leq B$. Note that in this case $t = O(B/\gamma)$, where $\gamma = 1/\operatorname{poly}(n)$ is the (additive) precision value.

▶ **Lemma 6.** *Let $n > k \geq 0$ be integers, $p \in (0, \infty)$ be a real and (x, y) be the input points. Assume that the polynomial $f(X) = \sum_{i=0}^{k} a_i X^i$ satisfies $\mathcal{E}_p^{(k)}(x, y) = \mathcal{E}_p(x, y, a)$. Then, for every $0 \leq i \leq k$, $|a_i| \leq 6n^{1/p} y_{\max} / \min(1, x_{\min}^k)$, where $y_{\max} = \max_i |y_i|$ and $x_{\min} = \min_i |x_i|$.*

By applying a random shift to the x values we may ensure that the numerator is $\Omega(1)$ and we may subsequently assume that $B = \operatorname{poly}(n)$. For constant k, this ensures that the post-processing step is polynomial in n. In the remainder of this section we show that this dependence on n can be made poly-logarithmic when k is constant and $p \geq 1$.

3.2.1 Poly-logarithmic post processing for constant k:

We start by defining the family of functions $h^j : \mathbb{R}^j \to \mathbb{R}$ for $j = 1, \ldots, k+1$:

$$h^j(a_k, \ldots, a_{k-j+1}) = \min_{a_0, \ldots, a_{k-j}} \sum_{i=1}^{n} \left| y_i - \sum_{m=0}^{k} a_m x_i^m \right|^p \quad \text{and} \quad h^0 = \mathcal{E}_p(x, y) \ .$$

In other words, h^j is the smallest interpolation error that can be achieved when the j highest coefficients are fixed. We first note that h^j is convex.

▶ **Lemma 7.** *For any $p \geq 1$, $j \in \{0, 1, \ldots, k\}$ and $a_k, \ldots a_{k-j+2} \in \mathbb{R}$, the function $h(x) = h^j(a_k, \ldots, a_{k-j+2}, x)$ is convex.*

```
$Evaluate[j, a_k, \ldots, a_{k-j+1}]$
1. If $j == k+1$, return a $(1+\epsilon)$ approximation based on the sketches.
2. Initialize $a \leftarrow -B$ and $b \leftarrow B$ where $B$ upper bounds the magnitude of all coefficients.
3. Repeat
   a. For $x \leftarrow a + \frac{b-a}{4}$, $y \leftarrow a + \frac{b-a}{2}$, and $z \leftarrow a + \frac{3(b-a)}{4}$
         $e_x \leftarrow Evaluate[j+1, a_k, \ldots, a_{k-j+2}, x]$
         $e_y \leftarrow Evaluate[j+1, a_k, \ldots, a_{k-j+2}, y]$
         $e_z \leftarrow Evaluate[j+1, a_k, \ldots, a_{k-j+2}, z]$
   b. If $e_y < e_z/(1+\epsilon)$ then $b \leftarrow z$ and repeat
   c. If $e_y < e_x/(1+\epsilon)$ then $a \leftarrow x$ and repeat
4. Until $e_y \geq \min(e_x, e_z)/(1+\epsilon)$ or $|b-a| < \gamma$
5. Return $e_y$
```

Figure 1 The Evaluate Algorithm

To find the minimum value of a convex function $h(\cdot)$ in the range $[a, b]$, a natural approach would evaluate h at a few intermediate points, e.g., $a < x < y < z < b$, and recurse on the appropriate subinterval of $[a, b]$ based on the intermediate valuations. If $h(y) \leq h(z)$, we deduce that the minimum lies in the range $[a, z]$ and if $h(x) \geq h(y)$ we deduce that the minimum lies in the range $[x, b]$. Note that one of the above cases must apply since h is convex. If x, y, z are equally spaced in the interval, after $O(\log n)$ iterations we can determine the value that minimizes h.

As a warm-up to the main algorithm of this section, we next present a $O(\log^k n)$ pass algorithm. The algorithm is based on the recursion:

$$h^j(a_k, \ldots, a_{k-j+2}, a_{k-j+1}) = \min_a \left(h^{j+1}(a_k, \ldots, a_{k-j+2}, a_{k-j+1}, a)\right) .$$

We can evaluate h^k for a given $a_k, \ldots, a_1$ in $O(\log n)$ as described above. By appealing to the above recurrence, we can then determine h^{k-1} in $O(\log^2 n)$ passes: we minimize h^{k-1} for a given $a_k, \ldots, a_2$ by performing the quaternary search to find a such that $h^{k-1}(a_k, \ldots, a_2) = h^k(a_k, \ldots, a_2, a)$. Since each evaluation of h^k requires $O(\log n)$ passes, it takes $O(\log^2 n)$ passes to evaluate h^{k-1} for a given $a_k, \ldots, a_2$. Continuing in this manner gives a $O(\log^k n)$ pass algorithm for evaluating h^0. This leads to the following theorem.

▶ **Theorem 8.** *Assume each coefficient may take only t different known values. Then there exists a $O(\log^k t)$ pass algorithm that computes $\mathcal{E}_p(x, y)$ exactly in $\tilde{O}(k)$ space and $O(1)$ per-item processing and $O(1)$ processing at the end of each pass.*

We next transform the multiple pass algorithm into a single pass algorithm where each of the evaluations performed in the quaternary search is computed using a single sketch of the data. In Figure 1, we present the algorithm *Evaluate* for approximating h^j. $Evaluate[j, a_k, \ldots, a_{k-j+1}]$ approximates $h^j(a_k, \ldots, a_{k-j+1})$ by minimizing a sequence of convex functions. Note that *Evaluate* is solely concerned with post-processing: while the points are being streamed it suffices to construct the appropriate sketches. Before we analyze the running time and accuracy of *Evaluate*, we need the following result.

In our algorithm it won't be possible to evaluate h exactly. However, the following lemma demonstrates that when the approximate evaluations become so close that it becomes impossible to evaluate pairwise comparisons, we have identified a sufficiently accurate approximation of the minimum.

▶ **Lemma 9.** *Let $h : [a,b] \to \mathbb{R}$ be a convex function and let $\tilde{h} : [a,b] \to \mathbb{R}$ satisfy $(1-\gamma) \leq h(x)/\tilde{h}(x) \leq (1+\gamma)$ for all $x \in \mathbb{R}$. Suppose for some a, b, $(1+\epsilon)\tilde{h}(y) \geq \max(\tilde{h}(z), \tilde{h}(x))$ where $x = a + (b-a)/4$, $y = a + (b-a)/2$, and $z = a + 3(b-a)/4$. Then*

$$(1+\epsilon)\tilde{h}(y) \geq \min_{x \in [a,b]} h(x) \geq \tilde{h}(y)/(1+8\epsilon) .$$

Proof. Without loss of generality assume that $h(x) \leq h(z)$. Note that $h(y) \leq h(z)$ because of convexity. We have to analyze the following two cases.

1. $h(x) \leq h(y)$: In this case the minimum value is at least

$$h(y) - 2(h(z) - h(y)) = 3h(y) - 2h(z) \geq \tilde{h}(y)[3/(1+\epsilon) - 2(1+\epsilon)^2] .$$

2. $h(y) \leq h(x)$: In this case the minimum value is at least

$$h(y) - (h(z) - h(y)) = 2h(y) - h(z) \geq \tilde{h}(y)[2/(1+\epsilon) - (1+\epsilon)^2] .$$

In either case, the minimum value is at least $\tilde{h}(y)/(1+8\epsilon)$ assuming $\epsilon < 1/15$. ◀

▶ **Theorem 10.** *The running time of Evaluate[0] is $O(\log^{k+1} n)$ and returns a value that satisfies $1/(1+O_k(\epsilon))^k \leq Evaluate[0]/\mathcal{E}_p(x,y) \leq (1+O_k(\epsilon))^k$.*

Using an appropriately rescaled ϵ when sketching the original points leads to a $(1+\epsilon)$ approximation using $O(\epsilon^{-2} \operatorname{polylog}(n))$ space and $O(\operatorname{polylog} n)$ update and post-processing time for constant k. We note that dependence on k is such that this approach is only practical for small values of k.

Proof of Theorem 10. For the running time, note that in each iteration the innermost loop is performed $O(\log n)$ times since $B = O(\operatorname{poly} n)$ and $\gamma = 1/\operatorname{poly}(n)$. The result follows because the depth of the recursion is at most $k+1$. The claim on the accuracy follows by induction on the depth and Lemma 9. ◀

3.3 Minimizing the maximum error

In this section, we consider the problem of finding coefficients a such that the maximum absolute error, $\mathcal{E}_\infty(x,y,a)$, is minimized. We will present two results. The first follows from a straight-forward observation and results in a constant pass algorithm that finds the error and the coefficients exactly. The second algorithm only uses a single pass but returns coefficients that minimize the maximum error up to a constant factor.

The first observation is that the problem can be expressed as a linear program in $O(k)$ variables,

$$\min \epsilon \quad \text{subject to} \quad -\epsilon \leq \sum_{j=0}^{k} a_j x_i^j - y_i \leq \epsilon \quad \forall i \in [n] .$$

Such a problem can be solved in constant passes in $O(n^\delta)$ space for any constant δ using the sub-linear time (for constant k) algorithm of Chan and Chen [2].

▶ **Theorem 11.** *It is possible to minimize $\mathcal{E}_\infty(x,y,a)$ in constant passes and $O(n^\delta)$ space for any constant δ.*

Our single pass algorithm is based on the following relationship between $\mathcal{E}_\infty$ and $\mathcal{E}_p$.

▶ **Proposition 12.** *For $p \geq \frac{\log n}{\log(1+\epsilon)}$, $\mathcal{E}_\infty^{(k)}(x,y) \leq \sqrt[p]{\mathcal{E}_p^{(k)}(x,y)} \leq (1+\epsilon)\mathcal{E}_\infty^{(k)}(x,y)$.*

Proof. The result follows because for any non-negative vector $z \in \mathbb{R}^n$ with $r = \max_i z_i$, $r \leq (\sum_{i\in[n]} z_i^p)^{1/p} \leq (nr^p)^{1/p} \leq (1+\epsilon)r$. ◀

In Section 3, we noted that it is possible to evaluate $\mathcal{E}_p^{(k)}(x,y)$ (and determine the corresponding polynomial) in $O(p^2k)$ space if p was even. Therefore, by choosing $p = 2\lceil(\log n)/(2\log(1+\epsilon))\rceil$ and appealing to Proposition 12, get the following theorem.

▶ **Theorem 13.** *$\mathcal{E}_\infty^{(k)}(x,y)$ can be $(1+\epsilon)$-approximated in a single pass with $O(\epsilon^{-2}k\,\mathrm{polylog}(n))$ space.*

References

1 P. A. Barker, F. A. Street-Perrott, M. J. Leng, P. B. Greenwood, D. L. Swain, R. A. Perrott, R. J. Telford, and K. J. Ficken. A 14,000-Year Oxygen Isotope Record from Diatom Silica in Two Alpine Lakes on Mt. Kenya. *Science*, 292(5525):2307–2310, 2001.

2 Timothy M. Chan and Eric Y. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007.

3 Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In *STOC*, pages 205–214, 2009.

4 Devdatt Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms.* Cambridge University Press, New York, NY, USA, 2009.

5 Ilya Dumer, Daniele Micciancio, and Madhu Sudan. Hardness of approximating the minimum distance of a linear code. *IEEE Transactions on Information Theory*, 49(1):22–37, 2003.

6 D. Feldman, M. Monemizadeh, C. Sohler, and D.P. Woodruff. Coresets and sketches for high dimensional subspace approximation problems. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, volume 2, 2010.

7 Sander Greenland. Dose-response and trend analysis in epidemiology: Alternatives to categorical analysis. *Epidemiology*, 6(4):356–365, 1995.

8 Venkatesan Guruswami and Alexander Vardy. Maximum-likelihood decoding of reed-solomon codes is np-hard. *IEEE Transactions on Information Theory*, 51(7):2249–2256, 2005.

9 Monika R. Henzinger, Prabhakar Raghavan, and Sridhar Rajagopalan. Computing on data streams. *External memory algorithms*, pages 107–118, 1999.

10 Piotr Indyk. Algorithms for dynamic geometric problems over data streams. In *STOC*, pages 373–380, 2004.

11 Piotr Indyk. Stable distributions, pseudorandom generators, embeddings, and data stream computation. *J. ACM*, 53(3):307–323, 2006.

12 Piotr Indyk. 6.895 Sketching, Streaming and Sub-linear Space Algorithms. *Lecture Notes*, 1, 2007.

13 Daniel M. Kane, Jelani Nelson, and David P. Woodruff. An optimal algorithm for the distinct elements problem. In *PODS*, pages 41–52, 2010.

14 S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1(2), 2005.

15 Atri Rudra and Steve Uurtamo. Data stream algorithms for codeword testing. In *ICALP*, 2010.

Spectral Sparsification in the Semi-Streaming Setting

Jonathan A. Kelner[1] **and Alex Levin**[2]

1 **Department of Mathematics**
Massachusetts Institute of Technology
Cambridge, MA 02139
kelner@mit.edu

2 **Department of Mathematics**
Massachusetts Institute of Technology
Cambridge, MA 02139
levin@mit.edu

Abstract

Let G be a graph with n vertices and m edges. A sparsifier of G is a sparse graph on the same vertex set approximating G in some natural way. It allows us to say useful things about G while considering much fewer than m edges. The strongest commonly-used notion of sparsification is spectral sparsification; H is a spectral sparsifier of G if the quadratic forms induced by the Laplacians of G and H approximate one another well. This notion is strictly stronger than the earlier concept of combinatorial sparsification.

In this paper, we consider a semi-streaming setting, where we have only $\tilde{O}(n)$ storage space, and we thus cannot keep all of G. In this case, maintaining a sparsifier instead gives us a useful approximation to G, allowing us to answer certain questions about the original graph without storing all of it. In this paper, we introduce an algorithm for constructing a spectral sparsifier of G with $O(n \log n/\epsilon^2)$ edges (where ϵ is a parameter measuring the quality of the sparsifier), taking $\tilde{O}(m)$ time and requiring only one pass over G. In addition, our algorithm has the property that it maintains at all times a valid sparsifier for the subgraph of G that we have received.

Our algorithm is natural and conceptually simple. As we read edges of G, we add them to the sparsifier H. Whenever H gets too big, we resparsify it in $\tilde{O}(n)$ time. Adding edges to a graph changes the structure of its sparsifier's restriction to the already existing edges. It would thus seem that the above procedure would cause errors to compound each time that we resparsify, and that we should need to either retain significantly more information or reexamine previously discarded edges in order to construct the new sparsifier. However, we show how to use the information contained in H to perform this resparsification using only the edges retained by earlier steps in nearly linear time.

1998 ACM Subject Classification G.2.2 [Discrete Mathematics]: Graph Theory—graph algorithms; G.3 [Probability and Statistics]: Probabilistic algorithms (including Monte Carlo)

Keywords and phrases Algorithms and data structures, graph algorithms, spectral graph theory, sub-linear space algorithms, spectral sparsification

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.440

1 Introduction

Graph sparsification is the task of finding a sparse approximation to a given (possibly weighted) graph G. Specifically, we seek a weighted graph H on the same vertex set as G

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 440–451
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

but with fewer edges, which approximates G in some respect. This enables one to approximately solve problems on G while performing calculations on a much simpler graph, which often gives substantially faster running times. This problem has been extensively studied by many researchers, and it has recently become a very active area of investigation as a result of its connections with numerous topics, including the construction of fast solvers for symmetric diagonally dominant linear systems (see, e.g., [11, 5]).

Benczúr and Karger [3] introduced the notion of combinatorial sparsification and proved that any graph G on n vertices has a combinatorial sparsifier H with $O(n \log n/\epsilon^2)$ edges. This H preserves the total weight of the edges crossing any cut of G up to a multiplicative $1 \pm \epsilon$ factor. (Note that this condition will require H to be weighted even if G is unweighted.) Moreover, the authors provided an efficient algorithm for computing H by sampling the edges of G according to certain very carefully chosen probabilities.

Recently, Spielman and Teng [10] defined the notion of spectral sparsification. A graph H is a $1 \pm \epsilon$ spectral sparsifier of G if it preserves the quadratic form induced by the graph Laplacian to within a multiplicative $1 \pm \epsilon$ factor. This definition is strictly stronger than combinatorial sparsification. In a remarkable paper, Spielman and Srivastava [9] showed that by sampling each edge with probability proportional to its effective resistance, adding it in with a certain weight, and taking $O(n \log n/\epsilon^2)$ independent samples in this manner, one obtains a $1 \pm \epsilon$ spectral sparsifier of G with high probability.[1] Furthermore, Spielman and Srivastava showed how to approximate *all* of the effective resistances to within a $1 \pm \eta$ factor in time $\tilde{O}(m/\eta^2)$, where m is the number of edges of G; using these approximate values allows one to construct a sparsifier with only a constant blowup in the number of edges. This leads to a nearly linear (specifically $\tilde{O}(m/\epsilon^2)$) time algorithm for computing spectral sparsifiers.

Having a sparsifier allows us to approximately answer numerous algorithmic questions about a dense graph without doing as laborious a calculation. For example, we can approximate the values of cuts, the effective resistances between pairs of vertices, and many properties of the behavior of random walks on the graph. As such, sparsifiers can be used to give faster approximate algorithms to numerous problems; Benczúr and Karger did this in their paper. Additionally, in a setting where we do not have room to store a given dense graph in its entirety, or perhaps can only access it through slow memory, a sparsifier, which might be small enough to store in fast memory, can be a useful proxy.

This, along with the growing need to process extremely large graphs, leads us to ask if we can also construct the sparsifier for a graph G even if we have much less work space than it would take to represent G. In particular, we consider the semi-streaming model, where we have $\tilde{O}(n)$ memory available. The algorithm of [9] requires access to the entire graph G, and thus, does not conform to the model. In this paper, we show how to build a sparsifier almost as quickly as with the original Spielman-Srivastava algorithm, while using at most $\tilde{O}(n)$ space and taking only one pass over G.

In our analysis, we will consider a more general dynamic setting, where we start with a graph G and its sparsifier H, and, as G is constantly updated, we want to maintain a $1 \pm \epsilon$ approximation to the current graph. Specifically, we consider the case of adding edges to G. (Setting the initial graphs G and H to be empty graphs on the vertex set V, we obtain the semi-streaming case discussed above; for details, see Section 3.7.) It is not hard to see that as

[1] Spielman and Srivastava actually showed that this holds with probability at least 1/2 using Rudelson's sampling theorem [7], but a straightforward application of a stronger concentration of measure result [12, Corollary 4] yields the high probability claim.

we add edges to G, by adding in those same edges to H, we get the desired approximation of G. Unfortunately, as we keep doing this, our sparsifier will contain increasingly many edges and may eventually become too large for our needs. Thus we will need to resample, to produce a sparsifier of smaller size. We show how to periodically do this resampling very fast, leading to amortized nearly constant update time per edge added to G. The resampling algorithm relies on two main insights:

1. As we add new edges to G to produce a graph G', the effective resistances of the edges of G do not increase, and thus, neither does their probability of being selected for a sparsifier. Thus, if we can compute their new probabilities, we can rejection sample the edges in H and also appropriately sample the new edges to produce edges selected with the probability distributions from G', and hence a sparsifier of G'. Thus, we need not consider all the probabilities in G', but only those of edges in H and the added edges.
2. Since H with the new edges well-approximates G', we can use it to quickly estimate the effective resistances for the edges we need; this estimate turns out to be good enough.

On a high level, the key idea of our construction is that the original sparsifier already contains a great deal of information, which we can reuse to save time instead of building a sparsifier from scratch.

In addition to being a generalization of the semi-streaming problem, the dynamic setting has numerous applications. As [2] points out, by maintaining a sparsifier of a changing graph at all times, we will be able to perform certain calculations quickly whenever we need to know something about the current graph. Furthermore, in some cases, it might be that we have a family of graphs we can choose from, and all of them are variants on some base graph G (e.g., G with extra edges). In that instance, by sparsifying G, and using our procedure, one can produce sparsifiers of the other graphs relatively quickly.

Related work

The problem of graph sparsification in the semi-streaming setting was introduced by Ahn and Guha [2], and it was then further studied by Goel, Kapralov, and Khanna [4] (the latter of which is concurrent to and independent of the present paper). Ahn and Guha constructed combinatorial sparsifiers in the semi-streaming model. However, while the space complexity of their algorithm was $\tilde{O}(n)$, the running time was $\tilde{O}(mn)$, which is often too slow when the graphs are large. This is remedied by the present work, as well as by Goel, Kapralov, and Khanna, who obtain results that are similar to ours when one aims to construct combinatorial sparsifiers.

However, the graphs that we produce obey the strictly stronger constraints imposed by spectral sparsification. To our knowledge, ours is the first work to do this in the semi-streaming setting.

Furthermore, we believe that our algorithm is conceptually cleaner and simpler than that of [4], and our techniques are quite different from theirs. The algorithm set forth by Goel *et al.* inherently requires a logarithmic number of passes through the data, and they maintain a multi-level collection of graphs and partitions of graphs. They then, using an ingenious construction and careful analysis, find a way to implement this in a single pass. This results in a graph that has logarithmically more edges than necessary, which they then clean up at the end.

Our algorithm, on the other hand, operates inherently in a single pass. We simply add edges to our graph until it becomes large. When this occurs, we replace our graph with a sparser version still preserving the approximation guarantee and continue. By taking advantage of the stronger notion of sparsification that we are employing, and properly sparsifying and analyzing the probabilities, we are able to show that this simple algorithm produces the desired sparsifiers while requiring a nearly constant amount of amortized work per edge and maintaining at all times a graph with $\tilde{O}(n/\epsilon^2)$ edges.

Acknowledgments This work was partially supported by NSF grant CCF-0843915. The second author is supported by a National Science Foundation graduate fellowship.

2 Background

Throughout, let $G = (V, E)$ be a graph on n vertices $V = \{1, \ldots, n\}$ and m edges. The Laplacian of G is given by $L_G := A_G - D_G$, where A_G is the adjacency matrix of G and D_G is the diagonal matrix whose i^{th} diagonal entry is the degree of vertex i. Let L_G^+ be the (Moore-Penrose) pseudoinverse of L_G.

For $i \in V$, we denote by χ_i the $n \times 1$ characteristic vector of i (having a 1 at the ith position and 0's everywhere else). For an edge e of G with endpoints $i, j \in V$, we let b_e be the $n \times 1$ vector $\chi_i - \chi_j$ (where the choice of i or j to have the positive sign is made arbitrarily). Further, denote by B the $m \times n$ edge-vertex incidence matrix, with rows indexed by the edges and columns indexed by the vertices, and whose e^{th} row is b_e. It is a standard fact that the Laplacian can be decomposed as $L_G = \sum_{e \in G} b_e b_e^T$, or, equivalently, that $L_G = B^T B$.

Finally, we discuss some notions from electrical network theory that will be relevant. Consider the graph as an electric network of nodes (vertices) and wires (edges), where each edge has resistance of 1. The *effective resistance* between vertices i and j is the voltage difference that would be induced between i and j if one unit of current were put in i and taken out from j. It can be shown that this is precisely equal to $(\chi_i - \chi_j)^T L_G^+ (\chi_i - \chi_j)$. For an edge e of G, the effective resistance of e, denoted by R_e, is defined to be the effective resistance between the endpoints of e. Namely, we have $R_e = b_e^T L_G^+ b_e$. It is a standard fact that if G is connected, we have $\sum_{e \in G} R_e = n - 1$. The sum is $n - c - 1$ if G has c connected components.

2.1 Spectral sparsifiers

Fix notation as above. In what follows, we will write $A \preceq B$ for matrices A and B when $B - A$ is positive semidefinite.

▶ **Definition 1.** A $1 \pm \epsilon$ spectral sparsifier of G is a possibly weighted graph H such that the edge set of H is contained in that of G and

$$(1 - \epsilon) L_G \preceq L_H \preceq (1 + \epsilon) L_G. \tag{1}$$

In other words, for all $x \in \mathbb{R}^n$, we have

$$(1 - \epsilon) x^T L_G x \leq x^T L_H x \leq (1 + \epsilon) x^T L_G x.$$

Note that we have approximations for pseudoinverses as well. If H is a $1 \pm \epsilon$ sparsifier of G, it is the case that

$$\frac{1}{1 + \epsilon} L_G^+ \preceq L_H^+ \preceq \frac{1}{1 - \epsilon} L_G^+. \tag{2}$$

In particular, because the effective resistance between i and j is given by evaluating the quadratic form defined by the Laplacian pseudoinverse at $\chi_i - \chi_j$, we see that the effective resistances between any two nodes in G and H are the same up to a $1/(1 \pm \epsilon)$ factor.

```
Algorithm 1 Sparsify
Input: $G = (V, E)$
  Set $H$ to be the empty graph on $V$.
  for $i = 1$ to $N = O(n \log n/\epsilon^2)$ do
      Sample edge $e \in G$ with probability $p_e$ proportional to $R_e$, and add it in with
      weight $1/(Np_e)$ to $H$
  end for
```

In [9], Spielman and Srivastava considered Algorithm 1 for generating a sparsifier of G. They proved:

▶ **Theorem 2.** *Fix $\epsilon \geq 1/\sqrt{n}$. Then, Algorithm 1 produces a $1 \pm \epsilon$ sparsifier of G with high probability.*

Additionally, the authors were able to show how to approximate all the effective resistances in time $\tilde{O}(m/\epsilon^2)$. Up to log factors, this is optimal, since it takes $\Omega(m)$ time to even write down all of the effective resistances. This fast way of estimating resistances gives a nearly-linear time (in m) algorithm for sparsifying G.[2]

Notation

Before proceeding, we make a few remarks about notation. Let G be a graph with n vertices. Let Γ be another graph on the same vertex set as G. Then, $G + \Gamma$ is the graph given by adding the weights of the edges of Γ to the corresponding edges of G. In this paper, for the most part G and Γ will be unweighted graphs, and Γ will be edge-disjoint from G. In this case, $G + \Gamma$ represents the graph we get when we add the edges of Γ to G. The definition agrees with the previous one if we regard missing edges as having a weight of 0, and those that are in the graph as having a weight of 1.

For an edge e not in G, we denote $G + e$ the graph obtained by adding e to G.

For convenience, we define $q = q(n, \epsilon) = O(\log n/\epsilon^2)$ to be the quantity such that taking nq samples in the above algorithm (using exactly correct probabilities) gives us a $1 \pm \epsilon$ sparsifier H of G with probability at least $1 - n^{-d}$ (for some desired constant d determined at the outset).

Finally, we note that as prescribed by [9], when we add an edge to H multiple times, we just sum up the weights. In our presentation, it will be convenient to think instead of keeping parallel edges, so that our sparsifiers will have have exactly nq edges.

3 The dynamic update algorithm

Throughout, for notational convenience, we will consider the setting of adding new edges to an unweighted graph G without adding new vertices. It is straightforward to generalize to the case where we add vertices, or where the graph is weighted and we may increase the weights of existing edges as well as add new ones, provided that the weights are polynomially bounded.

[2] Throughout, we will use the terms “nearly constant” or “nearly linear” to mean constant or linear, up to poly-logarithmic factors. This terminology is fairly standard.

3.1 Setup

Initially, we assume that we are given access to the exact effective resistances when we need them to sample. We will later relax this requirement.

Suppose that G is a graph on n vertices and H is a $1 \pm \epsilon$ sparsifier of G with nq edges. (If G is the initial graph, we assume that H was produced using the sampling procedure in [9], given the correct effective resistances.) Let e be an edge not in G; then it is clear that $H + e$ is a $1 \pm \epsilon$ sparsifier of $G + e$. Indeed, we have

$$L_{G+e} = L_G + b_e b_e^T, \qquad L_{H+e} = L_H + b_e b_e^T,$$

whence the desired statement follows.

As we add edges to G, we can add those same edges to H, until the sparsifier gets too large, forcing us to resample. In this work, we say that this happens when it is of size Cnq for some constant $C > 1$ that we can choose at will.

We will formalize this situation as follows. Let $G = (V, E)$ be a graph, and let H be its $1 \pm \epsilon$ sparsifier with nq edges. Let Γ represent the added edges (i.e., it is a graph on V with edges exactly those that are added to G) such that $H_+ := H + \Gamma$ has Cnq edges. (Note that H_+ is a $1 \pm \epsilon$ sparsifier of $G' := G + \Gamma$.)

Because H_+ is large, we want to construct a sparsifier H' of G' of nq edges. We call this procedure *resparsification*. We would like this resparsification to take much less time than it would take to sample from scratch, namely $\tilde{O}(m/\epsilon^2)$. Sparsifying G' from scratch gives us an average update time of $\tilde{O}(m/n)$ per operation, which is $\tilde{O}(n)$ when G' is dense. We want a $\tilde{O}(1)$ amortized time instead. The key insight is to use the information already contained in H, which will allow us to sample edges from the correct distribution in time $\tilde{O}(n/\epsilon^2)$, leading to the desired bound.

The main observation is that when we add a new edge to G, the effective resistances of the other edges cannot increase. This is more or less clear from the physical model, and it can be proven rigorously as in [6, Lecture 9]. Further, the sum of the effective resistances of all of the edges cannot decrease. (If adding the edge reduces the number of connected components, this quantity increases, otherwise it stays the same.) Thus, the probabilities of choosing the edges in the sparsification procedure of [9] cannot increase.

In what follows, we let R_e (resp. R'_e) be the effective resistances across edge e in G (resp. G'), and p_e (resp. p'_e) be the probability of selecting those edges (i.e. $p_e = R_e / \sum_{f \in G} R_f$, and similarly for p'_e).

Consider Algorithm 2, which details the resparsification step in our simplified context.

Algorithm 2 Resparsification (knowing the correct probabilities)

```
Input: $H, \Gamma$        %$G$, the original graph, is not given
Output: $H'$, a $1 \pm \epsilon$ sparsifier of $G'$ with $nq$ edges
1: Compute the effective resistances in $G$ of all the edges of $H$, and from there the probabilities of selecting them.        % We will show how to do this later
2: Compute the effective resistances in $G'$ of all the edges of $H_+$ (i.e. all the edges of $H$ and $\Gamma$) and the probabilities of selecting them.
3: for all edges $e$ of $H$ do
4:     With probability $p'_e/p_e$, add $e$ to $H'$ with weight $1/(nqp'_e)$
5:     Otherwise, pick an edge $f$ of $\Gamma$ with probability $p'_f / \left(\sum_{g \in \Gamma} p'_g\right)$ and add it to $H'$ with weight $1/(nqp'_f)$
6: end for
```

Note that the sampling procedure is well-defined, since $p'_e \leq p_e$. Further, we claim that all the edges are selected with exactly the correct probabilities, giving us a proper sample, so that H' is a $1 \pm \epsilon$ sparsifier of G' with high probability. Indeed, imagine selecting one edge for H' using a three-step process:

1. Select an edge from G with probability p_e.
2. Keep it with probability p'_e/p_e.
3. If you reject in Step 2, pick an edge f of Γ with probability $p'_f / \left(\sum_{g\in\Gamma} p'_g\right)$.

Note that you select edge $e \in G$ with probability p'_e, and the overall probability of rejecting in the second step is $1 - \sum_{e\in G} p'_e = \sum_{g\in\Gamma} p'_g$, hence you select f with probability p'_f, as it should be. By going through edges in H as in the algorithm and accepting them with the desired probability, we are effectively reusing the randomness we used to construct H. This, in turn, allows us to save time by only considering the probabilities for the Cnq edges of H_+, rather than for all the edges of G'.

The heart of algorithm then becomes correctly estimating the R_e and R'_e and from them, the p_e and p'_e.

3.2 Estimating effective resistances

Unfortunately, we are not able to exactly compute the effective resistances (and hence selection probabilities) quickly enough, so we will have to estimate them. We know that if our estimate of each probability is guaranteed to be at least $1/\alpha$ of the correct value (for some fixed $\alpha > 1$), then αnq samples are enough to get a $1 \pm \epsilon$ sparsifier with high probability (see [8, Corollary 6], where this fact is proven implicitly). We will be able to provide this guarantee for $\alpha \ll 2$, so if we let H have $2nq$ edges (and take $2nq$ samples for computing H'), the same claims about the sparsification quality of H and H' will hold.

With that in mind, we note that our method is closely related to the one in [9] for calculating the effective resistances quickly. We first recapitulate the ideas in that paper, and then we describe our modifications.

Recall that

$$R_e = b_e^T L_G^+ b_e = b_e^T L_G^+ L_G L_G^+ b_e = b_e^T L_G^+ B^T B L_G^+ b_e = \left\| B L_G^+ b_e \right\|^2 .$$

Thus, if $e = (i, j)$, then R_e is the squared distance between the ith and jth columns of the $m \times n$ matrix BL_G^+. To compute this would be too slow, so Spielman and Srivastava make the following approximations:

1. Since the effective resistances are given by distances between $n \times 1$ vectors, we can apply the Johnson-Lindenstrauss Theorem to randomly project the vectors onto a smaller dimensional subspace such that all the distances stay roughly the same with high probability [1].
   Specifically, for $k = O(\log n/\epsilon^2)$, if we choose a $k \times m$ matrix Q each of whose entries is $\pm 1/\sqrt{k}$ uniformly at random, then, with high probability over the choice of Q, we know that for every edge $e = (i, j)$ of G, the squared distance between the ith and jth column of the $k \times n$ matrix $Z = QBL_G^+$ approximates R_e to within a $1 \pm \epsilon$ factor. Furthermore, computing QB takes $O(km) = \tilde{O}(m/\epsilon^2)$ time, since B has only $2m$ entries. (We remark that the ϵ in the above can be the same as the parameter measuring the quality of the sparsifier.)
2. Computing L_G^+ is costly, so instead, we will approximate it by approximately solving linear systems in L_G. Each solution gives us an approximation of a row of Z, and takes $O(m \log(1/\delta))$ time (since L_G has $O(m)$ entries), using the Spielman-Teng linear system

solver for symmetric diagonally dominant systems [11]. Here, δ is a parameter for the quality of approximation, which can be appropriately set. Since there are k rows, we need to solve $k \ll n$ linear systems. Let $\tilde{Z}$ be the matrix we obtain in this fashion.

Spielman and Srivastava show that if Z encodes all the effective resistances to within a $1 \pm \epsilon$ factor (which happens with high probability over the choice of projection matrix Q), then for

$$\delta \leq \frac{\epsilon}{3}\sqrt{\frac{2(1-\epsilon)}{(1+\epsilon)n^3}} \tag{3}$$

we have that $\tilde{Z}$ encodes the effective resistances to within a $(1 \pm \epsilon)^2$ factor [8, Lemma 9]. The running time of one instance of the linear system solver is thus $\tilde{O}(m \log(1/\epsilon))$, and since there are at most k of them, the total time for computing $\tilde{Z}$ is $\tilde{O}(m/\epsilon^2)$.

For our purposes, we need to compute a matrix encoding approximations to effective resistances in faster time, namely $\tilde{O}(n/\epsilon^2)$. Now, because we already have sparsifiers (H and H_+) for G and G' of size $2nq$ and $2Cnq$ respectively, we can use their Laplacians in the place of L_G for estimating the R_e and R'_e respectively. With high probability, $Q_H W_H^{1/2} B_H L_H^+$ (resp. $Q_{H_+} W_{H_+}^{1/2} B_{H_+} L_{H_+}^+$) encodes the effective resistances of edges in H (resp. H_+) to within $1 \pm \epsilon$. Here, B_H is the edge-vertex incidence matrix of H, such that $L_H = B_H^T W_H B_H$ for W_H the matrix whose diagonal entries are the weighted degrees of H; Q_H is a $k \times 2nq$ matrix with entries uniform from $\pm 1/\sqrt{k}$. The definitions of B_{H_+}, W_{H_+}, and Q_{H_+} are similar.

We can compute the $Q_H W_H^{1/2} B_H$ in $O(kqn) = \tilde{O}(n/\epsilon^2)$ time, and similarly for the matrix $Q_{H_+} W_{H_+}^{1/2} B_{H_+}$, up to a factor of C.

Thus, by running the linear system solver, we obtain matrices, call them $\tilde{Z}_H$ and $\tilde{Z}_{H_+}$, encoding the approximations to effective resistances in H and H_+.

Because the effective resistances in H and H_+ are within a $1/(1 \pm \epsilon)$ factor of those of G, in this manner, we approximate the effective resistances in G to within a factor of $(1 \pm 2\epsilon)^3$. Furthermore, solving the linear systems now takes $\tilde{O}(n/\epsilon^2)$ time, since the Laplacians have $\tilde{O}(n/\epsilon^2)$ entries (we are absorbing the constant C into the big-$\tilde{O}$ notation).

3.3 An alternate sparsification algorithm

We would now like to use the approximate effective resistances to run a variant of Algorithm 2, which simulates the random process in Algorithm 1. For technical reasons, in this setting, it is useful to consider the following variant of Algorithm 1, and simulate it instead:

Algorithm 3 Alternative sparsify

```
Input: $G$
Output: $H$, a $1 \pm \epsilon$ sparsifier of $G$ (with high probability)
  for all edges $e$ of $G$ do
        for $i$ from 1 to $N := O(n \log^2 n/\epsilon^2)$ do      %Run this loop implicitly
              With probability $p_e = R_e/(n-1)$ add $e$ to $H$ with weight $1/(Np_e)$      %$R_e$
              is the effective resistance of $e$ in $G$.
        end for
  end for
  return $H$
```

This algorithm produces a $1 \pm \epsilon$ sparsifier H of G with high probability. Moreover, the number of edges in H is tightly concentrated around $O(N)$. If we underestimate each p_e by at most an α factor, and use these estimates to run the algorithm, we will have to increase N by the same α factor to get the high probability claim.

The inner loop is run implicitly; for each edge, given p_e, it is easy to determine how many samples of the edge will be taken, as this follows a binomial distribution. In particular, once we know the probability, the total running time of the algorithm is proportional to the total number of edges we select, which is $O(N)$ with high probability.

Note that the sparsifiers we get as a result have $O(\log n)$ more edges than the sparsifiers produced by Algorithm 1. We need the extra factor for a technical reason, and from now on, will assume that all the sparsifiers we deal with are of size $O(n \log^2 n/\epsilon^2)$.

As before, if we add multiple copies of an edge to H, we treat them as parallel edges in what follows.

3.4 Putting it all together

Now we are ready to show the final algorithm. So, let G be a graph and H its $1 \pm \epsilon$ sparsifier generated according to Algorithm 3, with $2N$ rather than N steps in the inner loop, where, from now on, $N := O(n \log^2 n/\epsilon^2)$. The sparsifier H has $O(N)$ edges with high probability. Let the $\tilde{p}_e$ be the estimates of the probabilities of edges in G, which were used to generate H. As before, Γ will represent the new edges (of which there will now be $O(n \log^2 n/\epsilon^2)$), $G' := G + \Gamma$, and $H_+ := H + \Gamma$. Denote by $\tilde{p}'_e$ the probabilities of edges in G', computed using H_+, as described previously.

Consider Algorithm 4.

Algorithm 4 Resparsification

```
Input: $H, \Gamma$, as well as the $\tilde{p}_e$ for every edge $e \in H$.
Output: $H'$, a $1 \pm \epsilon$ sparsifier of $G'$ with $O(n \log^2 n/\epsilon^2)$ edges with high probability, as well
    as $\tilde{p}'_e$ for every edge $e \in H'$.
Estimate the effective resistances in $G'$ of all the edges of $H_+$.
For $e \in H_+$, let $\tilde{p}'_e = \tilde{R}'_e/(n-1)$        %Good approximation to true $p_e$
for each edge $e$ of $H$ do
    $\tilde{p}'_e \leftarrow \min(\tilde{p}_e, \tilde{p}'_e)$
end for
for all edges $e$ of $H$ do
    Keep $e$ with probability $\tilde{p}'_e/\tilde{p}_e$ and add it to $H'$ with weight $1/(2\tilde{p}'_e N)$.
end for
for all edges $e$ of $\Gamma$ do
    for $i$ from 1 to $2N$ do        %Do this loop implicitly
        With probability $\tilde{p}'_e$ put $e$ into $H'$ with weight $1/(2\tilde{p}'_e N)$
    end for
end for
return $H'$ and the $\tilde{p}'_e$ for $e \in H'$.
```

It is not hard to see that this algorithm simulates the random process for sparsifying G' using Algorithm 3. (Again, we reuse the randomness used to generate H.) We can see that if we keep adding edges and resparsifying, we will put edges into the resulting sparsifier with exactly the desired probability, up to the modification in Step 4. The probability is over the randomness of the entire algorithm up to the current step.

We remark that we need Step 4 since approximation errors might cause the estimate of the probability of an edge to go up after we have added Γ, even though the true probabilities should go down. For rejection sampling to simulate the proper probability distributions, the probabilities have to be non-increasing. We show that the change does not in fact hurt our construction.

Indeed, suppose p_e and p'_e are the true probabilities of e in G and G' respectively, and assume that $\tilde{p}_e \geq p_e/\alpha$ and $\tilde{p}'_e \geq p'_e/\alpha$ for some α (which is much smaller than 2). We must have $p_e \geq p'_e$, hence $\tilde{p}_e \geq p_e/\alpha \geq p'_e/\alpha$, and hence $\min(\tilde{p}_e, \tilde{p}'_e)$ is at least as big as p'_e/α. Therefore, the samples we take suffice to guarantee that H' will be a $1 \pm \epsilon$ sparsifier with high probability.

Computing the matrices encoding the effective resistances takes $\tilde{O}(n/\epsilon^2)$ times. We only need to compute $\tilde{O}(n/\epsilon^2)$ effective resistances (since we do this only for edges in H and those in H_+). Sampling also takes $\tilde{O}(n/\epsilon^2)$ time. Since we resparsify every $\tilde{O}(n/\epsilon^2)$ steps, we conclude that the update procedure takes $\tilde{O}(1)$ steps per added edge.

This procedure works to give resparsified graphs with the correct approximation guarantee with high probability as long as the previous resparsified graph was a $1 \pm \epsilon$ sparsifier. (Note that by construction, the previous graph will always have edges drawn from a probability distribution that is close to correct; we need it to be a good sparsifier so that we can use it to quickly obtain good approximations to effective resistances.) We can union bound the probability of failure of any one of the resparsification steps (where we can include the event that the sparsifier ends up too big as a failure). The other potential place for error is when we apply the Johnson-Lindenstrauss theorem, but again, we have a high probability guarantee there, and are free to increase the number of rows k in the projection matrices Q_H and Q_{H_+} by a constant factor. Thus, by picking N and k to have a large enough constants at the outset, we can perform this procedure any desired polynomial number of times and be guaranteed that we always maintain a sparsifier with high probability.

By keeping careful track of the running times of the construction, we can prove:

▶ **Theorem 3.** *Our dynamic update algorithm takes $O(\log^4 n(\log\log n)^3 \log(1/\epsilon)/\epsilon^2)$ operations per added edge.*

Proof. The bottleneck in the algorithm is solving the linear systems in the resparsification step. Using the recent results of Koutis, Miller, and Peng [5], which give the best asymptotics known to date, the solution to each linear system for the Laplacian of a graph with $O(n\log^2 n/\epsilon^2)$ edges can be approximated in time $O(n\log^4 n(\log\log n)^3\log(1/\delta))$. Solving $O(\log n/\epsilon^2)$ linear systems with δ as in (3), requires time $O(n\log^6 n(\log\log n)^3\log(1/\epsilon)/\epsilon^4)$. Since we resparsify after adding $O(n\log^2 n/\epsilon^2)$ edges, the amortized cost is

$$O(\log^4 n(\log\log n)^3\log(1/\epsilon)/\epsilon^2)$$

per added edge, as claimed. ◀

3.5 Error-forgetfulness of the construction

Before concluding this section, we note one interesting property of our construction in Algorithm 4. Using H and H_+, which are approximations to G and G' respectively, we obtain estimates on effective resistances, which are slightly worse than those we would get had we used the full graphs G and G' (but allow us to do the computation much faster). Despite the approximations that we make, by taking twice as many samples as we would have needed had we known the true probabilities, we once again obtain a high-quality sparsifier (with high probability), allowing us to make the approximation all over. In other words, because we take enough samples, and do so intelligently, the errors we make in approximating the effective resistances do not propagate; the procedure has no memory for the approximations we made in the past.

Compare this to a more naïve approach to the problem of resparsifying. If we have G, G', H and H_+, defined as before, it is tempting to use Algorithm 1 to sparsify H_+ directly to a graph of size nq. Unfortunately, the resulting graph $\bar{H}$ is a $1 \pm \epsilon$ approximation of H_+, which is a $1 \pm \epsilon$ approximation of G', so $\bar{H}$ is only guaranteed to be a $(1 \pm \epsilon)^2 \approx 1 \pm 2\epsilon$ sparsifier of G'. In other words, the error propagates.

3.6 Straightforward generalizations

It is easy to generalize the above construction to the following cases. First, the construction goes through almost directly for the case of weighted graphs, where we are allowed to add weighted edges. For example, the probability of selecting an edge becomes the weight of that edge times its effective resistance. The weights with which we add sampled edges depend on their weights in G, so in order to do this properly, we should store the weights of the edges in the current sparsifier.

We can also consider operations where we increase the weight of an edge e of G by some amount w. In this case, we imagine adding an edge parallel to e and with weight w to G, and proceed as before (we add e' with weight w to H, and resparsify after some number of steps). The reason for considering parallel edges here is that while increasing the weight of an edge decreases the probabilities of other edges, it may increase the probability of that edge, which can stymie our construction. If we instead add an independent copy of the edge, all the arguments go through.

The only thing we have to be careful about is that in the weighted case, the value of δ in (3) depends on the ratio of the maximum weight to the minimum weight in the graph. If this is always bounded by some polynomial in n, then we need to add at most a factor of $\log n$ to the running time of the linear system solver, and hence of the overall algorithm.

Secondly, we can envision adding vertices as well as edges to G. Adding a vertex and connecting it by an edge to some existing vertex does not affect the effective resistances of the other edges, and it does not increase the number of connected components in the graph. Hence, once again, the probability of existing edges can only decrease, and we can use the same arguments. Here, by adding vertices, we increase the number of times we need to sample in the inner loop of Algorithm 3 in order to get a $1 \pm \epsilon$ approximation guarantee. If we have an upper bound on the number of vertices we will end up with, we can ensure that we take enough samples from the outset.

3.7 The semi-streaming setting

The dynamic algorithm described above goes through almost unchanged in the semi-streaming case (where we start with the empty graph). After adding the first $2CN$ edges (where $N = O(n \log^2 n/\epsilon^2)$), we use Algorithm 4 (with H set to the empty graph and Γ set to the current graph, and $2N$ iterations in the inner loop), giving us a $1 \pm \epsilon$ approximation to the current graph, containing of $2N$ edges in expectation. The number of edges is in fact tightly concentrated around this expectation. Then we continue as before, adding edges and resparsifying every $2CN$ steps.

For our algorithm to be valid in the semi-streaming model, we only need to prove that it requires $\tilde{O}(n/\epsilon^2)$ work space. But this is immediate, since, with high probability, we will only deal with graphs of $\tilde{O}(n/\epsilon^2)$ edges throughout the run.

If we would like to end up with a sparsifier containing $O(n \log n/\epsilon^2)$ edges, we can run Algorithm 1 on the output, which will change the final error guarantee from $1 \pm \epsilon$ to $(1 \pm \epsilon)^2$.

4 Conclusions and future work

We have presented an algorithm for maintaining a sparsifier of a growing graph, such that the average time is $\tilde{O}(1)$ for each added edge. The main idea is a resampling procedure that uses information in the existing sparsifier to construct a new one very quickly. Our construction is robust and holds relatively unchanged for several natural variants. An interesting question left open by our work is whether similar results could be obtained in a dynamic model that permits the removal of edges as well. While this is somewhat unnatural in the semi-streaming setting, it is a very reasonable goal in the dynamic setting where one aims to maintain a sparsifier for a graph that is changing over time.

References

1 Dimitris Achlioptas. Database-friendly random projections. In *PODS '01: Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, pages 274–281, New York, NY, USA, 2001. ACM.

2 Kook Jin Ahn and Sudipto Guha. Graph sparsification in the semi-streaming model. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming: Part II*, ICALP '09, pages 328–338, Berlin, Heidelberg, 2009. Springer-Verlag.

3 András A. Benczúr and David R. Karger. Approximating *s*-*t* minimum cuts in $O(n^2)$ time. In *STOC '96: Proceedings of the Twenty-Eighth Annual ACM symposium on Theory of Computing*, pages 47–55, New York, NY, USA, 1996. ACM.

4 A. Goel, M. Kapralov, and S. Khanna. Graph sparsification via refinement sampling. arXiv: 1004.4915 [cs.DS].

5 Ioannis Koutis, Gary L. Miller, and Richard Peng. Approaching optimality for solving SDD systems. In *FOCS '10: Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science*, 2010.

6 Gregory F. Lawler and Lester N. Coyle. *Lectures on contemporary probability*, volume 2 of *Student Mathematical Library.* American Mathematical Society, Providence, RI, 1999.

7 M. Rudelson. Random vectors in the isotropic position. *J. Funct. Anal.*, 164(1):60–72, 1999.

8 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. arXiv: 0803.0929v4 [cs.DS].

9 Daniel A. Spielman and Nikhil Srivastava. Graph sparsification by effective resistances. In *STOC '08: Proceedings of the 40th Annual ACM symposium on Theory of Computing*, pages 563–568, New York, NY, USA, 2008. ACM.

10 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. arXiv: 0808.4134 [cs.DS].

11 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *STOC '04: Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing*, pages 81–90, New York, NY, USA, 2004. ACM.

12 Roman Vershynin. A note on sums of independent random matrices after Ahlswede-Winter. `http://www.umich.edu/~romanv/teaching/reading-group/ahlswede-winter.pdf`.

Solovay functions and K-triviality *

Laurent Bienvenu[1], Wolfgang Merkle[2], and André Nies[3]

1 LIAFA – CNRS & Université de Paris 7
Case 7014, 75205 Paris Cedex 13, France
laurent.bienvenu@liafa.jussieu.fr

2 Institut für Informatik, Universität Heidelberg
Im Neuenheimer Feld 294, D-69120 Heidelberg, Germany
merkle@math.uni-heidelberg.de

3 University of Auckland
Private Bag 92019, Auckland, New Zealand
andre@cs.auckland.ac.nz

Abstract

As part of his groundbreaking work on algorithmic randomness, Solovay demonstrated in the 1970s the remarkable fact that there are computable upper bounds of prefix-free Kolmogorov complexity K that are tight on infinitely many values (up to an additive constant). Such computable upper bounds are called Solovay functions. Recent work of Bienvenu and Downey [STACS 2009, LIPIcs 3, pp 147-158] indicates that Solovay functions are deeply connected with central concepts of algorithmic randomness such as Ω numbers, K-triviality, and Martin-Löf randomness.

In what follows, among other results we answer two open problems posed by Bienvenu and Downey about the definition of K-triviality and about the Gács-Miller-Yu characterization of Martin-Löf randomness. The former defines a sequence A to be K-trivial if $K(A\restriction_n) \leq^+ K(n)$, the latter asserts that a sequence A is Martin-Löf random iff $C(A\restriction_n) \geq^+ n - K(n)$. So both involve the noncomputable function K. As our main results we show that in both cases $K(n)$ can be equivalently replaced by any Solovay function, and, what is more, that among all computable functions such a replacement is possible exactly for the Solovay functions. Moreover, similar statements hold for the larger class of all right-c.e. in place of the computable functions. These full characterizations, besides having significant theoretical interest on their own, will be useful as tools when working with K-trivial and Martin-Löf random sequences.

1998 ACM Subject Classification F.1.1, F.4.1

Keywords and phrases Algorithmic randomness, Kolmogorov complexity, K-triviality

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.452

1 Introduction

1.1 Algorithmic randomness and Kolmogorov complexity

The goal of the theory of algorithmic randomness is to give a formal meaning to the notion of “random object”. For finite discrete objects, such as finite binary sequences or strings, this was achieved by Solomonoff, Kolmogorov and Chaitin via the notion nowadays known as *Kolmogorov complexity*, where then a string is said to be random if it is incompressible in the sense of having roughly maximum Kolmogorov complexity. As usual, for a string w we

* Wolfgang Merkle was partially supported by the Deutsche Forschungsgemeinschaft (DFG) under grant ME 1806/3-1. André Nies was partially supported by the Marsden Fund of New Zealand under grant no. 08-UOA-187.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 452–463
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

distinguish the plain Kolmogorov complexity $C(w)$ and its prefix-free variant $K(w)$. Both are defined as the length of the least description of w with respect to some fixed additively optimal Turing machine U. That is, we ask for the length of a shortest string p such that $U(p) = w$; however, for the prefix-free variant we restrict attention to Turing machines with prefix-free domain [5, 10].

For infinite objects, such as infinite binary sequences (or sequences, for short), various notions of randomness have been proposed and studied. This extensive study led to a consensus that the "best" notion of randomness is *Martin-Löf randomness*, mainly because in many respects, Martin-Löf randomness is well-behaved, in that the main properties of Martin-Löf random sequences do match our intuition of what random sequences should look like. Moreover, the concept of Martin-Löf randomness is robust in the sense that it admits various equivalent definitions that are all natural and intuitively meaningful. For example, Martin-Löf random sequences can be characterized as the sequences that are unpredictable in the sense that certain effectively approximable betting games cannot win on these sequence. In the early 1970s, another important characterization in terms of incompressibility was found by Schnorr — and independently, in a slightly different form, by Levin — which asserts that for any sequence A,

$$A \text{ is Martin-Löf random} \Leftrightarrow K(A\restriction_n) \geq^+ n \tag{1}$$

where K is the prefix-free Kolmogorov complexity and $A\restriction_n$ denotes the prefix of A of length n (here the notation $\geq^+$ means "greater or equal up to a constant additive term that does not depend on the variable n", for formal definitions and more detailed explanations of this and other notation see Section 1.3).

Solovay functions

Bienvenu and Merkle [2] observed that the incompressibility characterization (1) of Martin-Löf randomness remains valid in case the function K is replaced by a suitable computable function f:

$$A \text{ is Martin-Löf random} \Leftrightarrow \left[f(A\restriction_n) \geq^+ n\right], \tag{2}$$

where it is easy to see that in addition the function f can be chosen to be an upper bound for K. Continuing this line of research, Bienvenu and Downey [1] considered "good" upper bounds for K, namely those that are computable, and are tight on infinitely many values. They called such bounds *Solovay functions*, as Solovay [14] was the first to show that such a function exists.

▶ **Definition 1.** A function $g\colon \mathbb{N} \to \mathbb{N}$ is an *upper bound for K (up to an additive constant)* if $K(n) \leq^+ g(n)$, and such a bound is *i.o. tight (up to an additive constant)* if for infinitely many n, $g(n) \leq^+ K(n)$. An i.o. tight upper bound g for K is a *Solovay function* in case g is computable. It is a *weak Solovay function* in case g is right-c.e.

Thus, K itself is a weak Solovay function. Among other results to be discussed below, Bienvenu and Downey demonstrated that any computable function g for which the equivalence (2) holds true must be a Solovay function.

▶ **Theorem 2** (Bienvenu-Downey). *Any computable upper bound f of K which satisfies the equivalence*

$$A \textit{ is Martin-Löf random} \Leftrightarrow \left[f(A\restriction_n) \geq^+ n\right]$$

is a Solovay function.

Left-c.e. reals and Ω numbers

Infinite binary sequences can be identified with binary expansions of reals in the unit interval in the canonical way. Then, a sequence is called *left-c.e.* if the corresponding real is the limit of an effectively given nondescending sequence of rational numbers. Martin-Löf random sequences that are left-c.e. exist, and have very interesting properties. For example, a left-c.e. real is Martin-Löf random if and only if it is Solovay complete, i.e., has only effective approximations from below that are as slow as any other effective approximation from below to any other left-c.e. real, up to a constant factor [5, 9]. Furthermore, Martin-Löf random left-c.e. reals can be characterized as the measures of the domains of universal Turing machines [5]. Letting

$$\Omega_g = \sum_{n\in\mathbb{N}} 2^{-g(n)}$$

one obtains as a variant of the latter result that a left-c.e. real is Martin-Löf random if and only if the real can be written in the form $\Omega_{\widetilde{K}}$ for some variant $\widetilde{K}$ of K obtained by using an alternate universal prefix-free Turing machine. In particular Ω_K is Martin-Löf random [5]. A full characterization of the computable functions g such that Ω_g is Martin-Löf random as the functions that are i.o. tight upper bounds for K was obtained by Bienvenu and Downey [1], and was extended to the class of right-c.e. functions by Hölzl et al. [8].

▶ **Theorem 3** (Bienvenu-Downey, Hölzl-Kräling-Merkle). *Let $g\colon \mathbb{N} \to \mathbb{N}$ be a right-c.e. function. Then g is a weak Solovay function if and only if Ω_g is a Martin-Löf random real. In particular, a computable function g is a Solovay function if and only if Ω_g is a Martin-Löf random real.*

Observe in this connection that by easy standard arguments, first, for any right-c.e. function g, the real Ω_g is finite if and only if g is an upper bound for K and second, a real is left-c.e. if and only if it can be written in the form Ω_g for some right-c.e. function g such that Ω_g is finite (where one exploits that left-c.e. reals have effective approximations from below by dyadic rationals, i.e., rationals of the form $p/2^q$ where $p, q \in \mathbb{N}$). Together, exactly the left-c.e. reals can be written in the form Ω_g for some right-c.e. upper bound g of K, and Theorem 3 states that the upper bound can be chosen to be i.o. tight if and only if the real is Martin-Löf random.

K-trivial sequences

From their incompressibility characterization, it can be seen that the Martin-Löf random sequences are those which have initial segments of roughly maximal Kolmogorov complexity. It is natural to ask which sequences have initial segments of *minimal* Kolmogorov complexity. It is immediate that any computable sequence has minimal Kolgomorov complexity because for such a sequence the prefix of any given length n will have the same Kolmorogov complexity as n itself, up to a fixed additive constant, which is then minimal since any code for the prefix can also be used as a code for n. Indeed, Chaitin [4] showed that the sequences A such that $C(A\restriction_n) \leq^+ C(n)$ are exactly the computable ones. On the other hand, this is not true any longer for the class of sequences A such that $K(A\restriction_n) \leq^+ K(n)$. While Chaitin [4] proved that any such sequence is computable from the halting problem, Solovay [14] was able to construct such a sequence that is noncomputable and computably enumerable. The class of such sequences was further studied by Downey, Hirschfeldt, Nies and Stephan [6, 12], who called these sequences *K-trivial*.

The K-trivial sequences turned out to have remarkable properties. Perhaps the most striking ones are that they can be characterized as the sequences that are low for Martin-Löf randomness, or, alternatively, as the sequences that are low for prefix-free Kolmogorov complexity. In other words, a sequence A is K-trivial if and only if Martin-Löf randomness relativized to A coincides with Martin-Löf randomness, if and only if the prefix-free Kolmogorov complexity relativized to A is within an additive constant of the unrelativized one. There are many more interesting results about K-trivial sequences. We refer the reader to the books by Downey and Hirschfeldt [5], and by Nies [13].

In Section 2 we will argue that in the definition of the notion of K-trivial, the upper bound $K(n)$ can be equivalently replaced by any weak Solovay function, and that in fact the ability to do so characterizes the Solovay functions and the weak Solovay functions. A preliminary result in this direction was obtained by Bienvenu and Downey [1], who showed that K-triviality can be characterized via some particular Solovay function.

▶ **Theorem 4** (Bienvenu-Downey). *There exists a Solovay function g such that for all A,*

$$A \text{ is } K\text{-trivial} \Leftrightarrow \left[K(A\restriction_n) \leq^+ g(n)\right]$$

The Gács-Miller-Yu Theorem

In view of the incompressibility characterization of Martin-Löf randomness in terms of prefix-free Kolmogorov complexity, it is suggestive to ask whether a similar characterization in terms of plain Kolmogorov complexity is possible. A first result in this direction was obtained by Gács [7] using conditional plain Kolmogorov complexity. He showed that for any sequence A,

$$A \text{ is Martin-Löf random} \Leftrightarrow \left[C(A\restriction_n |n) \geq^+ n - K(n)\right].$$

Much later, Miller and Yu [11] were able to show that this equivalence remains true when conditional plain Kolmogorov complexity is replaced by its unconditional counterpart. That is, for any sequence A,

$$A \text{ is Martin-Löf random} \Leftrightarrow \left[C(A\restriction_n) \geq^+ n - K(n)\right].$$

At the same time Miller and Yu showed that in addition the equivalence remains valid in case the term $K(n)$ is replaced by a suitable computable function g (a variation of the original Solovay function built by Solovay), which yields their celebrated characterization of Martin-Löf randomness based solely on plain Kolmogorov complexity: for some computable function g and for any sequence A,

$$A \text{ is Martin-Löf random} \Longleftrightarrow \left[C(A\restriction_n) \geq^+ n - g(n)\right].$$

For a simplified proof of their result see Bienvenu et al. [3].

1.2 Overview

By results discussed above, and by many other results not mentioned here, prefix-free Kolmogorov complexity is one of the most central notions in algorithmic randomness, and is indeed closely related to many other fundamental concepts in this area. In particular, as discussed above, the following assertions all become true in case we let g be equal to prefix-free Kolmogorov complexity K.

(i) The real Ω_g is Martin-Löf random.
(ii) A sequence A is K-trivial if and only if $K(A\restriction_n) \leq^+ g(n)$.

(iii) A sequence A is Martin-Löf random if and only if $C(A\restriction_n) \geq^+ n - g(n)$.
However, known results suggest that these close relations might not just hold for prefix-free Kolmogorov complexity but also for Solovay functions and weak Solovay functions in general. As stated above, the first assertion is true for a right c.e. function g if and only if g is a weak Solovay function [8]. Hence, as a special case, the first assertion is true for a computable function g if and only if g is a Solovay function [1]. For the two other assertions, on the other hand, it is only known that the second assertion is true for some Solovay function g [1], and, by the aforementioned result of Miller and Yu, that the third assertion holds true for some computable function g, while any function of the latter type must be a Solovay function [1].

In the present paper, we will investigate the question of which functions g make the second and third assertion true. Similar to the first assertion, we obtain a full characterization in the sense that the second as well as the third assertion is true for a right-c.e. function g if and only if g is a weak Solovay function, hence, is true for a computable function g if and only if g is a Solovay function.

▶ Remark. The result of Bienvenu and Downey that any computable upper bound f of K which satisfies the equivalence

$$A \text{ is Martin-Löf random} \Leftrightarrow \left[f(A\restriction_n) \geq^+ n\right]$$

must be a Solovay function does not extend to a characterization of Solovay functions, i.e., there are Solovay functions for which this equivalence is wrong. Indeed one can easily construct a Solovay function which is tight only on highly compressible sequences: take a Solovay function g, and define f by $f(0^n) = g(n)$ for all n, and $f(\sigma) = 3|\sigma|$ for all the other strings σ. It is clear that f is a Solovay function, but for $A = 10000\ldots$, one has $f(A\restriction_n) =^+ 3n \geq^+ n$, hence f does not characterize Martin-Löf randomness.

1.3 Notation

Here we gather some notation that will be used throughout the paper. A (binary) string is a finite sequence over the alphabet $\{0,1\}$. The set of all strings is denote by $\{0,1\}^*$, while $\{0,1\}^n$ and $\{0,1\}^{\leq n}$ denote the set of strings of length n and of length at most n, respectively. Strings are identified with natural numbers via the order isomorphism that takes the length-lexicographical order on strings to the usual order on $\mathbb{N} = \{0,1,\ldots\}$, for example, the empty string λ is identified with the natural number 0. Sequence refers to an infinite binary sequence, unless explicitly stated otherwise, and the set of sequences is denoted by $\{0,1\}^\omega$. For a sequence A, we write $A = A(0)A(1)\ldots$ and the prefix of A of length i is denoted by $A\restriction_i = A(0)\ldots A(i-1)$.

For a string σ, the *cylinder* $[\sigma]$ is the set of sequences A such that σ is a prefix of A. If S is a *set* of strings, we write $[S]$ for the set of sequences having some prefix in S, i.e. $S = \bigcup_{\sigma \in S}[\sigma]$. When we talk about *measure* on the space $\{0,1\}^\omega$ of sequences, we mean Lebesgue measure μ, which is the probability measure one gets when each bit of a sequence is chosen at random with probability $(1/2, 1/2)$ independently of all the other bits.

For functions f and g defined on some domain D such as the set of all strings or all natural numbers, the notation $f(n) \leq^+ g(n)$ means that there is some constant c such that for all $n \in D$ we have $f(n) \leq f(n) + c$, and $f(n) \geq^+ g(n)$ and $f(n) =^+ g(n)$ are defined likewise. Observe that this notation comprises a universal quantifier that ranges over D, hence it is slight abuse of notation, though straightforward, to extend to statements to statements such as "$f(n) \leq^+ g(n)$ holds for all n in some subset D_0 of D".

Plain Kolmogorov complexity is denoted by C, and its prefix-free variant by K; for definitions and further explanations we refer to the literature [5, 13, 10]. Kolmogorov complexity (plain or prefix-free) is defined on the set of finite string, but as usual we also apply it to other objects (integers, rational numbers, pairs of strings, etc.) as long as they can be encoded into finite strings in a computable way.

A function $f : D \to \mathbb{R}$ is right-c.e. (a.k.a. *approximable or semi-computable from above*) if there exists a computable function $F : D \times \mathbb{N} \to \mathbb{Q}$ such that for all $x \in D$, the values $F(x,0), F(x,1), \dots$ are nonincreasing and converge to $f(x)$ (the value $F(x,t)$ is called the approximation of $f(x)$ at stage t and is often denoted by $f_t(x)$ when the choice of a particular F is irrelevant in the argument). The plain and prefix-free variants of Kolmogorov complexity are examples of right-c.e. functions.

A bounded request set (a.k.a. Kraft-Chaitin set) is a computably enumerable set W of pairs (σ, n) of a string σ and a natural number n such that $\sum_{(\sigma,n)\in W} 2^{-n}$ is finite (enumerating a pair (σ, n) into a request set is often said to incur a *cost* of 2^{-n}; the request set being bounded if the total cost is finite). Having such a set, the *Kraft-Chaitin theorem* [5, 10, 13] asserts that for all $(\sigma, n) \in W$, one has $K(\sigma) \leq^+ n$.

2 K-triviality and Solovay functions

In this section, we prove that for any right-c.e. function g the equivalence

$$A \text{ is } K\text{-trivial} \iff [K(A\restriction_n) \leq^+ g(n)] \tag{3}$$

holds if and only if g is a weak Solovay function, i.e., if and only if g is an i.o. tight upper bound for K. Hence, in particular, for computable g, the equivalence (3) holds if and only if g is a Solovay function. Note that any function g that satisfies equivalence (3) must already be an upper bound of K, since $K(A\restriction_n)$ is always greater or equal to $K(n)$, up to an additive constant.

2.1 Solovay functions characterize K-triviality

We begin with the first part of the equivalence result, namely that K-triviality is characterized by weak Solovay functions and thus, in particular, by Solovay functions.

▶ **Theorem 5.** *Let g be a weak Solovay function. If $K(A\restriction_n) \leq^+ g(n)$, then A is K-trivial.*

As mentioned earlier (Theorem 4), this was proven by Bienvenu and Downey for a particular Solovay function, actually the one originally built by Solovay, which we call g_S. Their proof involved the construction of a bounded request set (or Kraft-Chaitin set), a standard technique to ensure the K-triviality of a sequence. However, it relied on the particular properties of the function g_S. We now show that given any weak Solovay function h and a sequence A such that $K(A\restriction_n) \leq^+ h(n)$, one can construct a bounded request set that ensures $K(A\restriction_n) \leq^+ g_S(n)$, hence proving the K-triviality of A. This is achieved by the following technical proposition, which will guarantee that building a bounded request set to ensure $K(A\restriction_n) \leq^+ g_S(n)$ does not "cost more" (in a specific sense to be explained below) than building a bounded request set to ensure $K(A\restriction_n) \leq^+ h(n)$.

▶ **Lemma 6.** *Let g be a Solovay function, and h a weak Solovay function. There exists a positive constant c and a computable partition of $\mathbb{N}$ into intervals $(I_n)_{n\in\mathbb{N}}$ such that for all n*

$$2^{-g(n)} \leq 2^c \sum_{i \in I_n} 2^{-h(i)}$$

Proof. We design a procedure which uniformly in k tries to construct a partition $(I_n^{(k)})_{n\in\mathbb{N}}$ such that $2^{-g(n)} \leq 2^k \sum_{i\in I_n} 2^{-h(i)}$. The procedure goes as follows:

For n from 0 to ∞ do

(1) Let $s(k,n) \in \mathbb{N}$ be the first integer which does not belong to one of the previously constructed intervals $I_j^{(k)}$ for $j < n$.

(2) Wait until we find some t large enough to have

$$\sum_{i=s(k,n)}^{t} 2^{-h_t(i)} \geq 2^{-k}2^{-g(n)}$$

(3) When this happens, we define $I_n^{(k)}$ to be $[s(k,n),t]$.

It is possible that for some (k,n) the procedure of parameter k waits at step 2 forever while executing the n-loop. When this happens, we have by construction:

$$\sum_{i\geq s(k,n)} 2^{-h(i)} \leq 2^{-k}2^{-g(n)}$$

Hence by the Kraft-Chaitin theorem, for all $i \geq s(k,n)$:

$$K(i) \leq^+ K(k,n,s(k,n)) + h(i) - k - g(n)$$

Since the construction is effective, $s(k,n)$ can be described via the pair (k,n) alone, hence $K(s(k,n)) \leq^+ K(k,n) \leq^+ K(n) + 2\log k$. This, together with the above inequality and the fact that $K(n) \leq^+ g(n)$ (because g is a Solovay function) yields for all $i \geq s(k,n)$:

$$K(i) \leq^+ h(i) - k + 2\log k$$

Now, recall that h is a weak Solovay function so $K(i) \geq^+ h(i)$ for infinitely many i. Therefore the above situation can only happen for a finite number of k. In other words, for all k large enough, the procedure never waits forever at step 2 and hence produces effectively a partition $(I_n^{(k)})_{n\in\mathbb{N}}$ of $\mathbb{N}$ into intervals such that for all n, and each $I_n^{(k)} = [s,t]$ we obtain as wanted

$$2^{-k}2^{-g(n)} \leq \sum_{i=s}^{t} 2^{-h_t(i)} \leq \sum_{i=s}^{t} 2^{-h(i)}.$$

◀

▶ **Corollary 7.** *For every weak Solovay function h, there exists a Solovay function $\tilde{h}$ such that $h \leq \tilde{h}$.*

Proof. Let h be a weak Solovay function and let g be any Solovay function. By Lemma 6, there exists a constant c and a computable partition $(I_n)_{n\in\mathbb{N}}$ of $\mathbb{N}$ into intervals such that for all n

$$2^{-g(n)} \leq 2^c \sum_{i\in I_n} 2^{-h(i)}$$

Let $\tilde{h} : \mathbb{N} \to \mathbb{N}$ be the function defined as follows. For a given i, let I_n be the interval to which i belongs, and set

$$\tilde{h}(i) = h_t(i) \text{ where } t \text{ is the least integer s.t. } 2^{-g(n)} \leq 2^c \sum_{i\in I_n} 2^{-h_t(i)}$$

It is clear that $\tilde{h}$ is computable and is an upper bound of h. Moreover, the sum

$$\sum_i 2^{-\tilde{h}(i)} = \sum_n \sum_{i \in I_n} 2^{-\tilde{h}(i)}$$

is random. Indeed, by construction for all n, $\sum_{i \in I_n} 2^{-\tilde{h}(i)} \geq^\times 2^{-g(n)}$. Hence $\sum_n 2^{-g(n)}$ is Solovay reducible to $\sum_i 2^{-\tilde{h}(i)}$ (the former being random, the latter must be too by the Kučera-Slaman theorem [9]). Therefore $\tilde{h}$ is a Solovay function. ◀

We are now ready to prove Theorem 5. Let h be a weak Solovay function, d a constant and A a sequence such that $K(A\restriction_n) \leq h(n)+d$ for all n. We want to prove that A is K-trivial. Since by Corollary 7 any weak Solovay function is dominated by a Solovay function, we only need to prove this theorem for h computable. We apply Lemma 6 to get a constant c and a computable partition of $\mathbb{N}$ into intervals $(I_n)_{n\in\mathbb{N}}$ such that for all n, $2^{-g_S(n)} \leq 2^c \sum_{i \in I_n} 2^{-h(i)}$. Without loss of generality, we also assume that for all n, $n < \min(I_n)$ (this can be ensured easily in the proof of Lemma 6).

We show that A is K-trivial by building a bounded request set. For all n and all strings σ of length n, we wait until we find an extension τ of σ whose length is $\max(I_n)$ and such that for all $i \in I_n$, some description of $\tau\restriction_i$ of length at most $h(i)+d$ is in the domain of $\mathbb{U}$ (by "description" we mean a string p such that $\mathbb{U}(p) = \tau\restriction_i$, where $\mathbb{U}$ is the universal prefix-free machine defining K). When (and if) this happens (we know when it does by computability of h), we enumerate a pair $(\sigma, g_S(n)+c+d)$ in our request set. The cost of this for us is $2^{-g_S(n)-c-d}$, which we can account against the cost for $\mathbb{U}$ to enumerate descriptions of $\tau\restriction_i$ as above, which is at least $\sum_{i\in I_n} 2^{-h(i)-d}$, which in turn is at least $2^{-g_S(n)-c-d}$ by construction of the intervals I_n. Hence, we never spend more than $\mathbb{U}$ does, which ensures that our request set is bounded. Now, by assumption on A, for every n, for every $i \in I_n$, the universal machine must issue a description of $A\restriction_i$ of length at most $h(i)+d$, hence some pair $(A\restriction_n, g_S(n)+c+d)$ enters our bounded request set at some point. Therefore, for all n, $K(A\restriction_n) \leq g_S(n)+c+d$. Applying Theorem 4, this shows that A is K-trivial.

2.2 K-triviality characterizes Solovay functions

We now prove that any right-c.e. function g that makes the equivalence

$$A \text{ is } K\text{-trivial} \iff [K(A\restriction_n) \leq^+ g(n)] \tag{4}$$

true is a weak Solovay function, and hence is a Solovay function in case g is computable. In the proof of our result, we need only to consider the case where g is an upper bound for K because otherwise the class of sequences A that satisfy the right-hand side of equivalence (4) is empty. We then prove the stronger fact that in the case g is a right-c.e. upper bound for K but is not a weak Solovay function, there are uncountably many sequences A such that $K(A\restriction_n) \leq^+ g(n)$. This is enough for our purposes, since there are only countably many K-trivial sequences (indeed, as we mentioned earlier, they are all computable in the halting problem).

▶ **Theorem 8.** *Let g be a right-c.e. function such that $K(n) \leq^+ g(n)$ but where g is not a weak Solovay function. Then the set $\{A \mid K(A\restriction_n) \leq^+ g(n)\}$ is uncountable.*

Proof. We will build an increasing sequence $a_1 < a_2 < a_3 < \ldots$ of integers such that any subset A of $\{a_1, a_2, a_3, \ldots\}$ satisfies $K(A\restriction_n) \leq^+ g(n)$.

The sequence is defined by induction (but not effectively), where we set $a_1 = 0$ and where we ensure by induction that for all k, for any subset B of the finite set $\{a_1, \ldots, a_k\}$ and for all $n \geq a_k$, for some constant d that does neither depend on B nor on k we have that

$$K(B\restriction_n) \leq g(n) - k + d. \tag{5}$$

This suffices to prove the desired result: let A be any subset of $\{a_1, a_2, a_3, \ldots\}$, and let n be some position. Let k be such that $a_k \leq n < a_{k+1}$. Let $B = A \cap \{a_1, \ldots, a_k\}$. Since $B\restriction_n = A\restriction_n$, one has by the above property $K(A\restriction_n) \leq^+ g(n) - k \leq^+ g(n)$.

We now explain the inductive definition of the sequence a_k. Suppose we have already defined $a_1, \ldots, a_k$ with the property (5). Let us choose c to be a very large integer, say $c > 2a_k + k + 1$. Consider the sum $\Omega_g = \sum_n 2^{-g(n)}$. By Theorem 3, this is not a random real as g is not a weak Solovay function. Hence, there exists a prefix σ of Ω_g such that $K(\sigma) \leq |\sigma| - c$. Let p be a shortest description for σ. Knowing p, one can effectively perform the following operations: first, retrieve $\sigma = \mathbb{U}(p)$; then, enumerate Ω_g from below and wait until it becomes larger than the real value $0.\sigma$ (treated as a real number written in binary) using the approximation of the values $g(n)$ from above; when this happens, let a_{k+1} be the least number m such that for all $i \geq m$, so far there has been no contribution to Ω_g by the value $g(i)$ (more precisely, via the approximation of these values from above). Since σ is a prefix of Ω_g, this means in particular that $\sum_{n \geq a_{k+1}} 2^{-g(i)}$ does not exceed $2^{-|\sigma|}$, so by the Kraft-Chaitin theorem, any integer $n \geq a_{k+1}$ can be described by p and some additional $g(n) - |\sigma|$ bits of information. Therefore, if $n \geq a_{k+1}$ and B is a subset of $\{a_1, \ldots, a_{k+1}\}$, then $B\restriction_n$ can be described in a prefix-free way by

- $B\restriction_{a_k}$,
- p (from which a_{k+1} can be retrieved),
- the single bit $B(a_{k+1})$,
- some additional $g(n) - |\sigma|$ bits.

Thus $K(B\restriction_n) \leq^+ 2a_k + |p| + 1 + g(n) - |\sigma| \leq^+ g(n) - (k+1)$ (using the fact that $c > 2a_k + k + 1$ and $|p| \leq |\sigma| - c$). This concludes the inductive step. ◀

3 Solovay functions and the Gács-Miller-Yu theorem

We now turn to the link between Solovay functions and the Gács-Miller-Yu theorem. Recall from the introduction that this theorem states that a sequence A is Martin-Löf random if and only if $C(A\restriction_n) \geq^+ n - K(n)$, and that moreover there exists a computable upper bound f of K such that A is Martin-Löf random if and only if $C(A\restriction_n) \geq^+ n - f(n)$. Bienvenu and Downey proved that any such function f must be a Solovay function. We now prove the converse, i.e. that *any* Solovay function makes this equivalence true, and the same is true for weak Solovay functions.

▶ **Theorem 9.** *Let g be a (weak) Solovay function. The following are equivalent.*
(i) $A \in \{0,1\}^\omega$ is Martin-Löf random.
(ii) $C(A\restriction_n) \geq^+ n - g(n)$.

We begin our proof with a combinatorial lemma.

▶ **Lemma 10.** *Let σ be a string. Let $I = [s,t]$ be a finite interval of integers with $s \geq |\sigma|$. Let $(a_i)_{i \in I}$ be a finite set of integers such that*

$$\sum_{i \in I} a_i 2^{-i} \geq 2^{-|\sigma|+1}.$$

Then, there exists a subset J of I and a finite set of strings S such that

(i) $[S] = [\sigma]$
(ii) *for all* $\tau \in S$, $|\tau| \in J$
(iii) *for all* $j \in J$, $|S \cap \{0,1\}^{\leq j}| \leq a_j$

Moreover, J *and* S *can be constructed effectively given* σ, I *and* $(a_i)_{i \in I}$.

Proof. We construct J and S via the following procedure. We initialize J and S to $\emptyset$. Now the procedure is as follows:

```
For all $i$ from $s$ to $t$ do
    If $a_i \leq |S|$ do nothing. Otherwise:
    (1) Put $i$ into $J$
    (2) Split $[\sigma] \setminus [S]$ into cylinders of measure $2^{-i}$. Let $T$ be the set of strings of length $i$ generating those cylinders.
    (3) Let $T'$ be the set containing the $c_i = a_i - |S|$ first strings of $T$ in the lexicographic order (if $c_i > |T|$ then let $T' = T$).
    (4) Enumerate all strings of $T'$ into $S$.
```

We now verify that this procedure works, i.e., that the algorithm is well-defined and that the set S we obtain after the t-loop is as wanted. First, notice that at the beginning of the i-loop, S contains only strings of length smaller than i, therefore $[S]$ can be split into cylinders of measure 2^{-i}. Since $|\sigma| \leq s \leq i$, this is also the case for $[\sigma]$, hence for $[\sigma] \setminus [S]$, so step (2) is well-defined. We also immediately see that the conditions (ii) and (iii) of the lemma are satisfied: indeed, we only enumerate strings of a given length i after enumerating i into J, and if we do so, we ensure that at the end of the i-loop, the cardinality of $S \cap \{0,1\}^{\leq i}$ is at most a_i. It remains to verify condition (i). First it is clear that $S \subseteq [\sigma]$ as we only enumerate cylinders that are contained in $[\sigma]$. Suppose that this inclusion is strict. Then, when running the above procedure, at step 3, we are never in the case where $c_i > |T|$, hence for all i, at the end of i-loop, we have $|S \cap \{0,1\}^{\leq i}| \geq a_i$, whether i is in J or not. Therefore, at the end of the procedure, we have:

$$\sum_{i=s}^{t} a_i 2^{-i} \leq \sum_{i=s}^{t} |S \cap \{0,1\}^{\leq i}| 2^{-i} \leq \sum_{i=s}^{t} \sum_{k=s}^{i} |S \cap \{0,1\}^{k}| 2^{-i} \leq \sum_{k=s}^{t} |S \cap \{0,1\}^{k}| \sum_{i=k}^{t} 2^{-i}$$
$$< \sum_{k=s}^{t} |S \cap \{0,1\}^{k}| 2^{-k+1} < 2\mu([S]) < 2\mu([\sigma]) < 2^{-|\sigma|+1}$$

and this contradicts the hypothesis of the lemma. ◀

Proof of Theorem 9. The part $(i) \rightarrow (ii)$ follows directly from the Gács-Miller-Yu theorem. We prove the converse. Let g be a weak Solovay function and $A \in \{0,1\}^{\omega}$ a sequence which is not Martin-Löf random. We shall prove that $C(A \upharpoonright_n) \leq n - g(n) - k$ for infinitely many n and arbitrarily large k. By Corollary 7, we can assume that g is computable. We further assume, for technical reasons which will become clear at the end of the proof, that for all i, either $g(i) \leq 2\log(i)$ or $g(i) = +\infty$. If it is not the case, replace g by the bigger function $\tilde{g}$ defined by $\tilde{g}(i) = g(i)$ if $g(i) \leq 2\log(i)$, and $\tilde{g}(i) = +\infty$ otherwise. Then we have:

$$\sum_{i} 2^{-\tilde{g}(i)} = \sum_{i} 2^{-g(i)} - \sum_{\substack{i \\ g(i) \geq 2\log i}} 2^{-g(i)}$$

the third sum is a computable real number as the i-th term is bounded by $1/i^2$. Thus $\sum_i 2^{-\tilde{g}(i)}$ is equal to a random real minus a computable real, hence is a random real and thus $\tilde{g}$ is still a Solovay function.

Now, let $(\mathcal{U}_k)_{k\in\mathbb{N}}$ be a Martin-Löf test covering A and such that $\mu(\mathcal{U}_k) \leq 2^{-2k-1}$ for all k. We design a procedure (P_k) which for all k tries to enumerate a set of strings S_k such that $[S_k] = \mathcal{U}_k$, with additional properties on the length of the strings it contains. We ensure that this procedure succeeds for almost all k by building an auxiliary test $\mathcal{V}_k$ which tests the randomness of $\sum_i 2^{-g(i)}$. The procedure (P_k) works as follows.

(1) Wait for a new cylinder $[\sigma]$ to be enumerated into $\mathcal{U}_k$.

(2) Choose a large integer s, say larger than 2^N with N larger than any integer mentioned so far in the construction (including k).

(3) Enumerate into $\mathcal{V}_k$ the real dyadic interval

$$\left[\sum_{i<s} 2^{-g(i)}, 2^{-|\sigma|+1+k} + \sum_{i<s} 2^{-g(i)}\right]$$

(4) Wait for a stage t such that

$$\sum_{i\leq t} 2^{-g(i)} > 2^{-|\sigma|+1+k} + \sum_{i<s} 2^{-g(i)}$$

(5) When this happens, we have $\sum_{i=s}^{t} 2^{-g(i)} > 2^{-|\sigma|+1+k}$. We then apply Lemma 10 with $a_i = 2^{i-g(i)-k}$ to get a finite set of strings S_k^σ and a finite set of integers J_k^σ such that $[S_k^\sigma] = [\sigma]$, for all $\tau \in S_k^\sigma$, $|\tau| \in J_k^\sigma$ and for all $j \in J_k^\sigma$, $|S^\sigma \cap \{0,1\}^{\leq j}| \leq a_j$. We then put all strings of S_k^σ into S_k and go back to step 1.

It is possible that for some k, (P_k) will at some point reach step 4 and wait there forever. We claim that this can only happen for finitely many k. Indeed, for a given k, we have $\mu(\mathcal{V}_k) \leq 2^{-k}$, because whenever a cylinder $[\sigma]$ enters $\mathcal{U}_k$ at step 1, an interval of length $2^{-|\sigma|+1+k}$ enters $\mathcal{V}_k$, hence $\mu(\mathcal{V}_k) \leq 2^{k+1}\mu(\mathcal{U}_k) \leq 2^{-k}$. Thus, $(\mathcal{V}_k)_{k\in\mathbb{N}}$ is a Martin-Löf test. Furthermore, if the procedure for S_k waits forever at some step 4, this precisely means that $\sum_i 2^{-g(i)}$ belongs to the dyadic interval which was put into $\mathcal{V}_k$ at step 3, and thus in that case $\sum_i 2^{-g(i)} \in \mathcal{V}_k$. Since $\sum_i 2^{-g(i)}$ is random, it can only belong to finitely many $\mathcal{V}_k$, hence for almost all k the procedure (P_k) never waits forever at step 4. In that case, the c.e. set S_k it builds does satisfy $[S_k] = \mathcal{U}_k$ by construction.

To finish the proof, let k be such that (P_k) succeeds. Since A is not Martin-Löf random, A belongs to $\mathcal{U}_k$, hence to $[S_k]$. This means that for some n, $A\restriction_n$ belongs to S_k. To describe $A\restriction_n$, it suffices to describe k (this can be done with $2\log k + O(1)$ bits), and its position inside S_k. For its position inside S_k, we simply describe the position of $A\restriction_n$ inside the S_k^σ it belongs to, when the latter is sorted in the length-lexicographic order. By construction of S_k^σ, n must be in J_k^σ (otherwise S_k^σ would be empty), and there are at most $a_n = 2^{n-g(n)-k}$ strings of length less than or equal to n in S_k^σ, and therefore we can specify the position of $A\restriction_n$ inside S_k^σ with $n - g(n) - k$ bits. Thus, our description of $A\restriction_n$ has total length $n - g(n) - k + 2\log k + O(1)$. Since k can be taken as large as wanted, this will be enough to prove the theorem, but one last thing we need to check is that this description is enough to retrieve $A\restriction_n$. Indeed, while we give the index of $A\restriction_n$ inside the S_k^σ it belongs to, we do not describe σ explicitly. However, σ can be found as follows. The description of $A\restriction_n$ we give has length $n - g(n) - k + 2\log k + O(1)$. By assumption, $g(n) \leq 2\log n$ and by construction of S_k^σ, $k \leq \log s \leq \log n$. Hence our description has length between $n - 3\log n + O(1)$ and $n + O(1)$. Hence the length of our description gives us n with logarithmic precision. This is enough to find the string σ such that $A\restriction_n$ belongs S_k^σ because by construction of S_k, if l is the length of some string in $S_k^{\sigma'}$ with $\sigma' \neq \sigma$, then either $2^l < n$ or $2^n < l$, and hence either $l < n - 3\log n$ or $n < l - 3\log l$. ◀

References

1 Laurent Bienvenu and Rodney Downey. Kolmogorov complexity and Solovay functions. In *Symposium on Theoretical Aspects of Computer Science (STACS 2009)*, volume 09001 of *Dagstuhl Seminar Proceedings*, pages 147–158, http://drops.dagstuhl.de/opus/volltexte/2009/1810, 2009. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany.

2 Laurent Bienvenu and Wolfgang Merkle. Reconciling data compression and Kolmogorov complexity. In *International Colloquium on Automata, Languages and Programming (ICALP 2007)*, volume 4596 of *Lecture Notes in Computer Science*, pages 643–654. Springer, 2007.

3 Laurent Bienvenu, Wolfgang Merkle, and Alexander Shen. A simple proof of Miller-Yu theorem. *Fundamenta Informaticae*, 83(1-2):21–24, 2008.

4 Gregory Chaitin. A theory of program size formally identical to information theory. *Journal of the Association for Computing Machinery*, 22:329–340, 1975.

5 Rodney Downey and Denis Hirschfeldt. *Algorithmic randomness and complexity.* Springer, 2010.

6 Rodney Downey, Denis Hirschfeldt, André Nies, and Frank Stephan. Trivial reals. In *Proceedings of the 7th and 8th Asian Logic Conferences*, pages 103–131. Singapore University Press, 2003.

7 Peter Gács. Exact expressions for some randomness tests. *Z. Math. Log. Grdl. M.*, 26:385–394, 1980.

8 Rupert Hölzl, Thorsten Kräling, and Wolfgang Merkle. Time-bounded Kolmogorov complexity and Solovay functions. In *Mathematical Foundations of Computer Science (MFCS 2009)*, volume 5734 of *Lecture Notes in Computer Science*, pages 392–402, 2009.

9 Antonin Kučera and Ted Slaman. Randomness and recursive enumerability. *SIAM Journal on Computing*, 31:199–211, 2001.

10 Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications.* Texts in Computer Science. Springer-Verlag, New York, 3rd edition, 2008.

11 Joseph Miller and Liang Yu. On initial segment complexity and degrees of randomness. *Transactions of the American Mathematical Society*, 360(6):3193–3210, 2008.

12 André Nies. Lowness properties and randomness. *Advances in Mathematics*, 197(1):274–305, 2005.

13 André Nies. *Computability and randomness.* Oxford Logic Guides. Oxford University Press, 2009.

14 Robert Solovay. Draft of a paper (or series of papers) on Chaitin's work. Unpublished notes, 215 pages, 1975.

Everywhere complex sequences and the probabilistic method *

Andrey Yu. Rumyantsev[1]

[1] **Moscow State University, Russia**

Abstract

The main subject of the paper is everywhere complex sequences. An everywhere complex sequence is a sequence that does not contain substrings of Kolmogorov complexity less than $\alpha n - O(1)$ where n is the length of the substring and α is a constant between 0 and 1.

First, we prove that no randomized algorithm can produce an everywhere complex sequence with positive probability.

On the other hand, for weaker notions of everywhere complex sequences the situation is different. For example, there is a probabilistic algorithm that produces (with probability 1) sequences whose substrings of length n have complexity $\sqrt{n} - O(1)$.

Finally, one may replace the complexity of a substring (in the definition of everywhere complex sequences) by its conditional complexity when the position is given. This gives a stronger notion of everywhere complex sequence, and no randomized algorithm can produce (with positive probability) such a sequence even if αn is replaced by $\sqrt{n}$, $\log^* n$ or any other monotone unbounded computable function.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Kolmogorov complexity, everywhere complex sequences, randomized algorithms, Medvedev reducibility, Muchnik reducibility

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.464

1 Introduction

The paper considers binary sequences with substrings of high Kolmogorov complexity. Kolmogorov complexity of a binary string is the minimal length of a program that produces this string. We refer the reader to [1] or [2] for the definition and basic properties of Kolmogorov complexity.

The Levin–Schnorr Theorem (see, e.g., [1]) characterizes randomness of a sequence in terms of complexity of its prefixes. It implies that a n-bit prefix of a Martin-Löf random sequence has complexity $n - O(1)$. (Technically, we should consider monotone or prefix complexity here; for plain complexity we have $n - O(\log n)$ bound, but in this paper logarithmic precision is enough.) So sequences with complex prefixes exist (and, moreover, fair coin tossing produces such a sequence with probability 1).

If we require all substrings (not only prefixes) to be complex, the situation changes. Random sequences no longer have this property, since every random sequence contains arbitrarily long groups of consecutive zeros (and these groups have very small complexity).

However, sequences with this property ("everywhere complex") still exist. The following Lemma (proved by Levin [3]) says that there exists a sequence where every substring has high

* Supported by RFBR 0901-00709a and NAFIT ANR-08-EMER-008 grants.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 464–471
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

complexity (though the condition is now weaker; the complexity is greater than $\alpha n - O(1)$ where n is the length and $0 < \alpha < 1$).

Here is the exact statement. Let $\omega([i, j))$ be a substring $\omega_i\omega_{i+1}\omega_{i+2}\dots\omega_{j-1}$ of a sequence ω; let $K(u)$ be the Kolmogorov complexity of a binary string u.

▶ **Lemma 1** (Levin). *Let α be a real number, $0 < \alpha < 1$. There exists a sequence ω such that*

$$\mathrm{K}(\omega([k, k + n))) \geq \alpha n - O(1).$$

for all natural numbers k and n.

Here the constant $O(1)$ may depend on α but not on n and k.

Levin's proof in [3] used complexity arguments: informally, we construct the sequence from left to right adding bit blocks; each new block should increase the complexity as much as possible.

Later it became clear that this lemma has a combinatorial meaning: if for every n some $2^{\alpha n}$ strings of length n are "forbidden", there exists an infinite sequence without long forbidden substrings. This combinatorial interpretation shows that the statement of the lemma (and even a stronger statement about subsequences, not only substrings) is a corollary of the Lovász local lemma (see [4, 5]). Recently two more proofs were suggested (by Joseph Miller [6] and Andrej Muchnik).

Before stating our results, let us mention the following slightly generalized version of Levin's lemma. Though not stated explicitly in [3], it can be proved by the same argument.

▶ **Lemma 2** (Levin, generalized). *Let α be a real number, $0 < \alpha < 1$. Then there exists a sequence ω such that*

$$\mathrm{K}(\omega([k, k + n)) \mid k, n) \geq \alpha n - O(1).$$

for all integers k, n.

Here $K(x|y)$ denotes conditional Kolmogorov complexity of a string x when y is given (i.e., the minimal length of a program that transforms y to x). The difference is that substrings are now complex with respect to their position and length (so, for example, the binary representation of k can not appear starting from position k). In combinatorial terms, we have different sets of forbidden substrings for different positions. (In fact, n is not important here since its complexity, $O(\log n)$, can be absorbed by changing α.)

One can ask how "constructive" the proofs of Levin's lemma and its variants could be. There are several different versions of this question. One may assume that the set of forbidden strings is decidable and ask whether there exists a computable sequence that avoids all sufficiently long forbidden strings. Miller's argument shows that this is indeed the case, though a similar question of 2D configurations (instead of 1D sequences, cf. [4]) is still open.

In this paper we consider a different version of this question and ask whether there exists a probabilistic algorithm that produces a sequence satisfying the statement of Levin's Lemma (or some version of it) with positive probability.

2 The results

We say that a sequence ω is α-*everywhere complex* if

$$\mathrm{K}(\omega([k, k + n))) \geq \alpha n - c$$

for some constant c and for all integers k and n.

▶ **Theorem 3.** *No probabilistic algorithm can produce with positive probability a sequence ω that is α-everywhere complex for some $\alpha \in (0,1)$.*

▶ **Theorem 4.** *Let $\sum_{i=0}^{\infty} a_i$ be a computable converging series of nonnegative rational numbers. There exists a probabilistic algorithm that produces with probability* 1 *some sequence ω such that*

$$\mathrm{K}(\omega([k,k+n))) \geq a_{[\log n]}n - c$$

for some c and for all k and n.

▶ **Theorem 5.** *No probabilistic algorithm can produce with positive probability a sequence ω with the following property: there exists a non-decreasing unbounded computable function $g\colon \mathbb{N} \to \mathbb{N}$ such that*

$$\mathrm{K}(\omega([k,k+n)) \mid k,n) \geq g(n)$$

for all k and n.

Theorem 3 and 4 complement each other: the first one says that α-everywhere complex sequences for a fixed $\alpha > 0$ (even very small) cannot be obtained by a probabilistic algorithm; the second one says that if we allow sublinear growth and replace the bound αn by $\sqrt{n}$ or $n/\log^2 n$, then the probabilistic algorithm exists. (There are intermediate cases where none of these theorems is applicable, say, $n/\log n$ bound; we do not know the answer for these cases.)

Theorem 5 says that Theorem 4 cannot be extended to the case of the generalized Levin lemma; here the answer is negative for any computable non-decreasing unbounded function.

3 Proof of Theorem 4

Let us start with the positive result.

Proof of theorem 4. The idea of the construction is simple. We fix some computable function $f\colon \mathbb{N} \to \mathbb{N}$ and then let $\omega_i = \tau_{f(i)}$ where τ_i is a sequence of random bits (recall that we construct a probabilistic algorithm that uses random bit generator).

In other words, we repeat the same random bit τ_j several times at the locations ω_i where $f(i) = j$. Why does this help? It allows us to convert bounds for the complexity of *prefixes* of τ into bounds for the complexity of *substrings* of ω. Indeed, if we have some substring of ω and some additional information that tell us where several first bits of τ are located in the substring, we can reconstruct a prefix of τ.

We now give more details. We may assume without loss of generality that n, the length of a substring, is large enough. We may also assume that n is a power of 2, i.e., that $n = 2^m$ for some m. Indeed, for every substring x we can consider its prefix x' whose length is the maximal power of 2 not exceeding the length of x. The bound for complexity of x' gives the same bound (up to a constant factor) for the complexity of x.

Consider the substring $\omega([k,k+2^m))$ for some k and m. We want it to contain all the bits from some prefix of τ, more specifically, the first $a_m 2^m$ bits $\tau_0,\ldots,\tau_{a_m 2^m - 1}$ of τ. (We may assume without loss of generality that $a_m 2^m$ is an integer.)

To achieve this, we put each of these bits at the positions that form an arithmetic progression with common difference 2^m. The first term of this progression will be smaller than its difference, and therefore each interval of length 2^m contains exactly one term of this progression.

In this way for a given m we occupy a_m-fraction of the entire space of indices (each progression has density $1/2^m$ and we have $a_m 2^m$ of them). So to have enough room for all

m we need that $\sum a_m \le 1$. This may be not the case at first, but we can start with large enough m_0 to make the tail small.

Technically, first we let $m = m_0$ and split $\mathbb{N}$ into 2^m arithmetic progressions with difference 2^m. (The first progression is formed by multiples of 2^m, the second is formed by numbers that are equal to 1 modulo 2^m, etc.) We use first $a_m 2^m$ of them for level m reserving the rest for higher levels. Then we switch to level $m = m_0 + 1$, splitting each remaining progression into two (even and odd terms), use some of them for level $m_0 + 1$, convert the rest into progressions with twice bigger difference for level $m_0 + 2$, etc. (Note that if in a progression the first term is less than its difference, the same is true for its two halves.)

This process continues indefinitely, since we assume that $a_{m_0} + a_{m_0+1} + \cdots \le 1$. Note that even if this sum is *strictly* less than 1, all natural numbers will be included in some of the progressions: indeed, at each step we cover the least uncovered yet number. So we have described a total computable function f (its construction depends on m_0, see below).

Now we translate lower bounds for complexity of prefixes of τ into bounds for complexity of substrings of ω: the substring $\omega([k, k + 2^m))$ contains first $a_m 2^m$ bits of τ (for $m \ge m_0$), and the positions of these bits can be reconstructed if we know $k \bmod 2^m$ and the function f. This additional information uses $O(m)$ bits (recall that $m_0 \le m$ and it determines f). So

$$\mathrm{K}(\omega([k, k + 2^m))) \ge \mathrm{K}(\tau([0, a_m 2^m))) - O(m) \ge a_m 2^m - O(m).$$

The last term $O(m)$ can be eliminated: increasing a_m by $O(m)/2^m$, and even more, say, by $m^2/2^m$, we do not affect the convergence. (The bounds presented are literally true for prefix complexity; plain complexity of prefixes of τ is a bit smaller but the difference again can be easily absorbed by a constant factor that does not affect the convergence.) ◀

4 Proof of Theorem 5

The proofs of Theorem 3 and Theorem 5 are based on the same idea. We start with proving Theorem 5 as it is simpler.

Proof of Theorem 3. Fix some probabilistic algorithm A. We need to prove that some property ("there exists a non-decreasing unbounded computable function g" such that $K(\omega([k, k + n))|k, n) \ge g(n)$ for all k and n") has probability 0 with respect to the output distribution of A. Since there are countably many computable functions g, it is enough to show that *for a given* g this happens with probability 0. So we assume that both A (probabilistic algorithm) and g (a computable monotone unbounded function) are fixed, and for a given $\varepsilon > 0$ prove that the property "$K(\omega([k, k + n))|k, n) \ge g(n)$ for all k and n" has probability smaller than ε.

Assume first that probabilistic algorithm A produce an infinite output sequence with probability 1, and therefore defines a computable probability distribution P_A on the Cantor space of infinite sequences.

Consider some n. First we prove that for large enough N it is possible to select one "forbidden" string of length n for each starting position $k = 0, 1, \ldots, N - 1$ in such a way that the event "output sequence avoids all the forbidden strings" (at the corresponding positions) has probability less than ε.

This can be proved in several different ways. For example, we can use the following probabilistic argument. Let us choose the forbidden strings randomly (independently with the random bits used by A). For every output sequence of A the probability that it avoids all randomly selected "forbidden" strings is $(1 - 2^{-n})^N$ which is less than ε if N is sufficiently large. Therefore, the overall probability of the event "output of A avoids all forbidden strings" (with respect to the product distribution) is less than ε. Now we use averaging in different order and conclude that there exists one sequence of N forbidden strings with the required property.

After the existence of such a sequence is proved, it can be found by exhaustive search (recall that P_A is computable). Let us agree that we use the first sequence with this property (in some search order) and estimate the complexity of forbidden strings when length n and position k are known. The value of N is a simple function of n and ε (which is fixed for now, as well as A), and we do not need any other information to construct forbidden strings. So their conditional complexity is bounded and is less than $g(n)$ for large enough n. So the probability that all the substrings in the output of A will have complexity greater than g(their length), is less than ε.

It remains to explain how to modify this argument for a general case, without the assumption that A generates infinite sequences with probability 1. Let us modify the function $N(\varepsilon, n)$ in such a way that $(1-2^{-n})^{N(n,\varepsilon)} < \varepsilon/2$. Consider the probability of the event "A generates a sequence of length $N(n,\varepsilon)+n$". If somebody gives us (in addition to n and ε) an approximation from below for this probability with error at most $\varepsilon/2$, we may enumerate A's output distribution on strings of length $N+n$ and stop when the lower bound is reached. Then we apply the argument above using this restricted distribution and show that for this restricted distribution the probability to avoid simple strings is less than $\varepsilon/2$, which gives ε-bound for the full distribution (since they differ at most by $\varepsilon/2$). It is important here that the missing information is of size $\log(1/\varepsilon)+O(1)$, so for a fixed ε we need $O(1)$ additional bits. ◀

5 Proof of Theorem 3

The proof of Theorem 3 is similar to the preceding one, but more technically involved. In the previous argument we were allowed to choose different forbidden strings for different positions, and it was enough to use one forbidden string for each position. Now we use the same set of forbidden strings for all positions, and the simple bound $(1-2^{-n})^N$ is replaced by the following lemma.

▶ **Lemma 6.** *Let $\alpha \in (0,1)$. For every $\varepsilon > 0$ there exist natural numbers n and N (with $n < N$) and random variables $\mathcal{A}_n, \mathcal{A}_{n+1}, \ldots, \mathcal{A}_N$ whose values are subsets of $\mathbb{B}^n, \mathbb{B}^{n+1}, \ldots, \mathbb{B}^N$ respectively, that have the following properties:*

(1) the size of subset $\mathcal{A}_i$ never exceeds $2^{\alpha i}$;

(2) for every binary string x of length N the probability of the event "for some $i \in \{n, \ldots, N\}$ some element of $\mathcal{A}_i$ is a substring of x" exceeds $1-\varepsilon$.

The number n can be chosen arbitrarily large.

(We again use the probabilistic argument; this lemma estimates the probability for every specific x and some auxiliary probability distribution; the output distribution of randomized algorithm A is not mentioned at all. Then we use this lemma to get an estimate for the combined distribution, and change the order of averaging to prove the existence of finite sets $A_n, \ldots, A_N$ with required properties.)

Proof. First let us consider the case $\alpha > 1/2$. Then we actually need only two lengths n and N, where $N \gg n$, all other lengths are not used and the corresponding random subsets can be empty. For length n, we consider a uniform distribution on all sets of size $2^{\alpha n}$; all these sets have equal probabilities to be a value of random variable $\mathcal{A}_n$. For length N the set $\mathcal{A}_N$ is chosen in some fixed way (no randomness), see below.

Assume that some string x of length N is fixed. There are two possibilities:

(**a**) there are at least $2^{n/2}$ different substrings of length n in x;

(**b**) there are less than $2^{n/2}$ different substrings.

In the first case (a) strings of length n play the main role. Let S be a set of n-bit strings that appear in x; it contains at least $2^{n/2}$ strings. The probability that the desired event does not happen does not exceed the probability of the following event: "making $2^{\alpha n}$ random choices among n-bit strings, we never get into S". (It is a bit smaller, since now we can choose the same string several times.) This probability is at most

$$(1-2^{-n/2})^{2^{\alpha n}} = (1-2^{-n/2})^{2^{n/2}2^{(\alpha-1/2)n}} \approx (1/e)^{2^{(\alpha-1/2)n}}$$

and converges to zero (rather fast) as $n \to \infty$.

In the second case (b) strings of length N come into play. We may assume that N is a multiple of n. Let us split x into blocks of size n. We know that x has some special property: there are at most $2^{n/2}$ different blocks. Note that for large N the number of strings with this special property is less than $2^{\alpha N}$. Indeed, to encode such a string x, we first list all the blocks that appear in x (this is a very long list, but its length is determined by n and does not depend on N), and then specify each block by its number in this list. In this way we need $N/2+O(1)$ bits (the number is half as long as the block itself) and this is less than αN for large N. So for such a large N we may include all strings with this property in A_N and get the desired effect with probability 1.

Now let us consider the case when $\alpha > 1/3$ (but can be less than $1/2$). Now we need three lengths $n_1 \ll n_2 \ll n_3$. We will use n_2 that is a multiple of n_1, and n_3 that is a multiple of n_2. For length n_1 we again consider a random set of $2^{\alpha n_1}$ strings of length n_1. It guarantees success if the string x contains at least $2^{(2/3)n_1}$ different blocks of length n_1.

Now we compile a list of possible blocks of size n_2 that are "simple", i.e., contain at most $2^{(2/3)n_1}$ different blocks of size n_1. The same argument as before shows that a simple block can be described by $(2/3)n_2 + O(1)$ bits, where $O(1)$ depends only on n_1. Now A_{n_2} is a random set of $2^{\alpha n_2}$ *simple* blocks of size n_2. Then the argument again splits into two sub-cases. (Recall that we assume now that x is made of simple blocks of size n_2.)

The first case happens when x contains more than $2^{n_2/3}$ different simple blocks. Then with high probability some block of x appears in A_{n_2}.

The second case happen when x contains less than $2^{n_2/3}$ different simple blocks. Then x can be encoded by the list of these blocks, and this requires $n_3/3+O(1)$ bits. So if n_3 is large enough (compared to n_2), all possibilities can be included in A_{n_3}, and this finishes the argument for $\alpha > 1/3$.

A similar argument with four layers works for $\alpha > 1/4$, etc. ◀

This lemma will be the main technical tool in the proof of Theorem 3. But first let us prove a purely probabilistic counterpart of Theorem 3 that is of independent interest.

▶ **Theorem 7.** *Let $\alpha \in (0,1)$. For every probability distribution P on Cantor space Ω, there exist sets $A_1, A_2, \ldots$ of binary strings such that*

(1) *the set A_n contains at most $2^{\alpha n}$ strings of length n;*

(2) *with P-probability 1 a random sequence has substrings in A_i for infinitely many i.*

The possible "philosophical" interpretation of this theorem: one cannot prove the existence of sequences that avoid almost all A_i by a direct application of the probabilistic method; something more delicate (e.g., Lovász local lemma) is needed.

Proof. Let us first consider sequences of some finite length N and the induced probability distribution on them. We claim that for every ε and for large enough N we can choose $A_1, \ldots, A_N$ in such a way that they satisfy (1) and P-probability to avoid them is less than ε.

To show this, consider the (independent) random distribution on strings of lengths $1, \ldots, N$ provided by the lemma. What is the probability that a random string avoids a random set (with respect to the product distribution of P and the distribution provided by the lemma)? Since for every fixed string the probability is less than ε (assuming N is large enough), the overall probability (the average) is less than ε. Changing the order of averaging, we see that for some $A_1, \ldots, A_N$ the corresponding P-probability is less than ε.

Note that in fact we do not need short strings; strings longer than any given n are enough (if N is large). So we can use this argument repeatedly with non-overlapping segments $[n_i, N_i]$ and ε_i decreasing fast (e.g., $\varepsilon_i = 2^{-i}$). Then for P-almost every sequence we get infinitely many violations. Moreover, since the series $\sum \varepsilon_i$ is converging, P-almost every sequence hits an A_j where $j \in [n_i, N_i]$ for all but finitely many i (Borel–Cantelli lemma). ◀

Now we are ready to prove the weak version of Theorem 3:

Let $\alpha \in (0,1)$. There is no randomized algorithm that produces α-everywhere complex sequences with probability 1.

(The difference with the full version is that here we have probability 1 instead of any positive probability and that the value of α is fixed.)

To prove this statement, let us consider the output distribution P of this algorithm. Since the algorithm produces an infinite sequence with probability 1, this distribution is a computable probability distribution on the Cantor space. This measure can be then used to effectively find sequences ε_i, n_i, N_i and sets A_j as described so that with P-probability 1 a random sequence hits an A_j where $j \in [n_i, N_i]$ for all but finitely many i. Since the sets A_j can be effectively computed and have at most $2^{\alpha j}$ elements, every element of A_j has complexity at most $\alpha j + O(\log j)$; the logarithmic term can be absorbed by a change in α.

This argument shows also that for every computable probability distribution P and every $\alpha \in (0,1)$ there exists a Martin-Löf random sequence with respect to P that is not α-everywhere complex. One more corollary: for every $\alpha \in (0,1)$ the (Medvedev-style) mass problem "produce an α-everywhere complex sequence" is not Medvedev (uniformly) reducible to the problem "produce a Martin-Löf random sequence".

It remains to make the last step to get the proof of Theorem 3.

Proof of Theorem 3. If the probability to get an everywhere complex sequence is positive, then for some α the probability to get an α-everywhere complex sequence for this specific α is positive. (Indeed, we may consider only rational α and use countable additivity.)

So we assume that some α is fixed and some probabilistic algorithm produces α-everywhere complex sequences with positive probability. We cannot apply the same argument as above. The problem is that the output of the algorithm (restricted to the first N bits) is a distribution on $\mathbb{B}^N$ that is not computable (the probability that at least N bits appear at the output, is only a lower semicomputable real). However, for applying our construction for some ε_i, it is enough to know the output distribution up to precision $\varepsilon_i/2$ (in terms of statistical distance), as explained in the proof of Theorem 5, we replace our distribution by its part, and the error is at most $\varepsilon/2$. For this we need only $\log(1/\varepsilon_i) + O(1)$ bits of advice, which can be made small compared to αn. ◀

Now we get a stronger statements for mass problems:

▶ **Theorem 8.** *The mass problem "produce an everywhere complex sequence" is not Muchnik (non-uniformly) reducible to the problem "produce a Martin-Löf random sequence".*

Proof. Indeed, imagine that for every random sequence there is some oracle machine that transforms it to an everywhere complex sequence. Since the set of oracle machines is countable, some of then should work for a set of random sequences that has positive measure, which contradicts Theorem 3. ◀

The author thanks Steven Simpson for asking the question, and Joseph Miller and Mushfeq Khan for the discussion and useful remarks.

References

1 Li M., Vitanyi P, *An Introduction to Kolmogorov Complexity and Its Applications*, 2nd ed. N.Y.: Springer, 1997.

2 Alexander Shen, Algorithmic Information Theory and Kolmogorov Complexity, December 2000, lecture notes. Published as Technical Report 2000-034, Uppsala University, `http://www.it.uu.se/research/publications/reports/2000-034`.

3 Bruno Durand, Leonid Levin, Alexander Shen, Complex tilings, Journal of Symbolic Logic, 73(2), 593–613. (Preliminary version: STOC 2001.) See also: `http://arXiv.org/abs/cs.CC/0107008`

4 Andrey Rumyantsev, Forbidden Substrings, Kolmogorov Complexity and Almost Periodic Sequences, STACS 2006, 23rd Annual Symposium on Theoretical Aspects of Computer Science, Marseille, France, February 23–25, 2006. Lecture Notes in Computer Science, 3884, Springer, 2006, p. 396–407.

5 Andrey Rumyantsev, Kolmogorov Complexity, Lovász Local Lemma and Critical Exponents. Computer Science in Russia, 2007, Lecture Notes in Computer Science, 4649, Springer, 2007, p. 349-355.

6 Joseph Miller, *Two notes on subshifts.* Available from `http://www.math.wisc.edu/~jmiller/downloads.html`

Online Scheduling with Interval Conflicts

Magnús M. Halldórsson*[1], Boaz Patt-Shamir†[2], and Dror Rawitz[2]

1 **School of Computer Science, Reykjavik University, Menntavegur 1, 101 Reykjavik, Iceland.**
`mmh@ru.is`

2 **School of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel.**
`{boaz, rawitz}@eng.tau.ac.il`

Abstract

In the problem of Scheduling with Interval Conflicts, there is a ground set of *items* indexed by integers, and the input is a collection of *conflicts*, each containing all the items whose index lies within some interval on the real line. Conflicts arrive in an online fashion. A scheduling algorithm must select, from each conflict, at most one survivor item, and the goal is to maximize the number (or weight) of items that survive all the conflicts they are involved in. We present a centralized deterministic online algorithm whose competitive ratio is $O(\lg \sigma)$, where σ is the size of the largest conflict. For the distributed setting, we present another deterministic algorithm whose competitive ratio is $2 \lg \sigma$, in the special *contiguous* case, in which the item indices constitute a contiguous interval of integers. Our upper bounds are complemented by two lower bounds: one that shows that even in the contiguous case, all deterministic algorithms (centralized or distributed) have competitive ratio $\Omega(\lg \sigma)$, and that in the non-contiguous case, no deterministic oblivious algorithm (i.e., a distributed algorithm that does not use communication) can have a bounded competitive ratio.

1998 ACM Subject Classification F.2.2 Analysis of Algorithms and Problem Complexity: Non-numerical Algorithms and Problems—Computations on discrete structures and Sequencing and scheduling; G.2.1 Discrete Mathematics: Combinatorics—Combinatorial algorithms

Keywords and phrases online scheduling, online set packing, interval conflicts, competitive analysis, compound tasks, distributed algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.472

1 Introduction

We study the following abstract problem, which we call *Scheduling with Interval Conflicts*. There is a universe U of n *items*, each with an integer identifier. The input is a collection $\mathcal{C}$ of *conflicts*, where each conflict $C \in \mathcal{C}$ is a set containing all the items of U within some interval on the real line. A conflict represents an event where the specified items compete for a resource that can be granted to only one item. Conflict resolution is carried out by a scheduling algorithm that decides which item survives: all other items in the conflict set are eliminated. The goal of the scheduling algorithm is to maximize the number (or weight) of items that survive *all* their conflicts.

Scheduling with Interval Conflicts arises naturally in some scenarios. One interpretation of the model is when we have a set of permanently-running stations that may interfere

* Supported in part by the Iceland Research Fund (grant 90032021).
† Supported in part by the Israel Science Foundation (grant 1372/09) and by a grant from Israel Ministry of Science and Technology.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 472–483
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

only with other neighboring stations, where the underlying metric space is a line, and the interference range in each direction may change in every step. In each step we need to choose a station that will win the current conflict, if any. The goal is to maximize the number of stations that never fail.

Another example for our model are tasks that must be processed by a few bounded-capacity servers located at different sites on the Internet. The tasks are sent to these servers in the same order, and due to varying congestion conditions in the network, they arrive at the servers with varying burstiness: for example, the input to server A may be such that at step t task i arrives, at step $t+1$ tasks $i+1, i+2, i+3$ arrive together, at step $t+2$ no task arrives etc. The input to another server B may exhibit a different burst structure, e.g., tasks i and $i+1$ arrive together, and tasks $i+2$ and $i+3$ arrive together. Assume that the servers can process only one task at a step, and tasks cannot be stored for later processing. Then a time step in which more than a single task arrives can be represented as an interval conflict. The main question in our model is which tasks to process and which to drop, so as to maximize the total number of tasks that receive all the processing they require.

Finally, consider multiple streams of data-frames (e.g., video frames) that need to be transmitted across the Internet. Since data frames are typically too large to fit in a single packet, the frames are broken into a number of packets, and reconstructed at the receiver. However, if a packet is lost in transit, its whole constituent frame (i.e., *item*) becomes useless. Interval conflicts arise if the streams pass through a congested router which can forward only one packet from each burst of packets that arrive together (all other packets are dropped).

Problem variants

In some cases, conflicts need to be resolved without knowledge of other conflicts (for example, if conflicts arrive in different locations, or if the conflict resolution protocol must be stateless). We call this variant *oblivious* (or *distributed*) scheduling. In some other cases, all previous conflicts and their outcomes are known to the algorithm when a new conflict arrives. We call this variant *sequential* (or *centralized*) scheduling. Note that both oblivious and sequential scheduling are *online*, i.e., no information about future conflicts is available to the algorithm (the offline variant of the problem is when all conflicts are given ahead of time).

An interesting special case of interval conflicts is when the universe of items contains no gaps, i.e., the items have identifiers $i_0, i_0+1, \ldots, i_0+n-1$ (in general, item identifiers are only required to be totally ordered). We refer to this as the *contiguous* case.

1.1 Our Contribution

In this paper we introduce and formalize the problem of Scheduling with Interval Conflict (abbreviated SIC below), and give deterministic online algorithms and lower bounds on the competitive ratio of deterministic algorithms. We start off with the special case of *contiguous* conflicts. It turns out that contiguous conflicts allow for an oblivious (and hence distributed) algorithm, guaranteeing competitive ratio of $O(\lg \sigma)$, where σ is the maximal number of items in a conflict. However, no competitive oblivious algorithm exists if item identifiers are not contiguous, as we show. We then give a sequential algorithm whose competitive ratio is also $O(\lg \sigma)$. The algorithm works also in the case of weighted items and non-contiguous item identifiers. Both algorithms are matched by a $\Omega(\lg \sigma)$-lower bound on the competitive ratio of any deterministic online algorithm, even sequential algorithms for unweighted contiguous SIC.

Several additional results are omitted for luck of space. One is a simple algorithm in the case of resource augmentation, that is 1-competitive when allowed to accept two items per conflict. Another is an oblivious $O(\lg \sigma/b)$-competitive algorithm for the generalized problem when b items may survive each conflict. Finally, we present an alternative sequential algorithm whose competitiveness is expressed in terms of the *depth* of the interval structure, where depth is defined to be the maximal number of conflicts that any single item is involved in.

1.2 Related Work

The offline version of our problem, finding a maximum subset of points with no two in a common interval, is easily solvable in polynomial time (see Section 2). A related minimization problem is finding the minimum number of points intersecting all intervals, or alternatively minimum clique partition. A 2-competitive online algorithm for the latter problem is given and shown to be the best possible in [6].

Note the unusual characteristic of our problem is that the solution only decreases as more of the input arrives. Little is known about online maximization problems of this sort; the only related result we are aware of is [3].

A different dual problem is the interval selection problem, where we seek a maximum cardinality subset of disjoint intervals. In the online version, the intervals that arrive over time must be irrevocably accepted or rejected. Randomized algorithms for different cases are known [9, 1, 2]; the result closest in spirit to ours is an $O(\log m)$-competitive algorithm (originally for call control on the line) [1], where m is the number of possible interval endpoints. In general, however, a $\Omega(n)$ lower bound holds for the competitive ratio of randomized algorithms [2], where n is the number of intervals. Interval selection can be seen as an instance of scheduling with conflicts, which has been studied extensively (see, e.g., the surveys of [7, 10]), but to the best of our knowledge, we are the first to consider online conflicts in the form of groups of consecutive items.

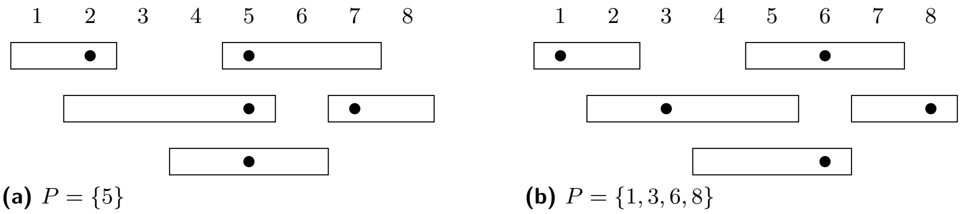
The problem of multi-packet frames (sketched above) was introduced in [8], where it is shown that if packet ordering is arbitrary (namely conflicts are not necessarily intervals), then the competitive ratio is $\Omega(\sigma)$ even for two-packet frames. A general framework that deals with transmission of multi-packet frames is described in [3]. The problem is modeled as an online version of Set Packing, nearly tight bounds of $\tilde{\Theta}(k\sqrt{\sigma})$ are proven on the competitive ratio of randomized algorithms for Online Set Packing, and a $\Omega(\sigma^{k-1})$ deterministic lower bound is shown, where k is the maximum size of a set and σ is the maximum number of sets that contain the same element. In our terms, it is assumed there that each item is involved in up to k conflicts, and conflicts need not be intervals.

1.3 Paper Organization

The remainder of this paper is organized as follows. In Section 2 we formalize the problem and present the basic arguments we use in analyzing our algorithms. We study oblivious algorithms in Section 3, and sequential algorithms are considered in Section 4. In Section 5 we prove a lower bound on the competitive ratio of online algorithms. Some concluding remarks are given in Section 6.

2 Preliminaries and Basic Argument

In this section we formalize the problem, define the concepts and notation we use, and present the basic argument we employ in the analysis of our algorithms.



(a) $P = \{5\}$

(b) $P = \{1, 3, 6, 8\}$

Figure 1 *An instance of* SIC *with two possible solutions. Rectangles represent conflicts, and dots represent items that were selected as survivors in conflicts. The instance contains eight items and five conflicts whose size is at most 4 (i.e., $\sigma = 4$). In Figure 1a (top), only one item survives all conflicts, while an optimal solution can have four such items (Figure 1b).*

2.1 Problem Statement and Notation

Scheduling with Interval Conflicts (abbreviated SIC) is defined as follows. There is a set U of n integer *items*. The input is a collection $\mathcal{C}$ of *conflicts*, where each conflict $C \in \mathcal{C}$ contains all items of U within some interval on the real line. Namely, $C = U \cap [\min(C), \max(C)]$. In this paper we also consider the *Contiguous Model*, where U is a set of consecutive integers. The size of the largest conflict is denoted by $\sigma_{\mathcal{C}}$, namely $\sigma_{\mathcal{C}} \overset{\text{def}}{=} \max\{|C| : C \in \mathcal{C}\}$ (the subscript is omitted when the instance is clear from the context). A feasible schedule is a set of items $P \subseteq U$ containing at most one item from any given conflict, i.e. $|P \cap C| \leq 1$ for every $C \in \mathcal{C}$. An item in P is said to be a *survivor* of its conflicts, while the other items were *eliminated*. If item i survives conflict $C \ni i$ by algorithm A, we say that A *delivers* i from C. The goal is to find a maximum cardinality feasible schedule, i.e. maximize the number of items surviving all their conflicts (see example in Figure 1).

In the *weighted* case, each item i has a real-valued weight $w(i) > 0$ and the objective is to find a maximum weight subset of weights satisfying the conflict constraints. For a set S of items, $w(S) \overset{\text{def}}{=} \sum_{i \in S} w(i)$.

We consider two models of algorithms. In the *oblivious* model, the selection of a survivor from a conflict is a function of that conflict only, which allows for distributed conflict resolution. In the *sequential* model, conflicts arrive over time, i.e., they are ordered as a sequence $C_1, C_2, \ldots$, and the resolution of conflict C_t may be a function of the full history $C_1, \ldots, C_t$.

We note that simple heuristics for SIC may perform poorly. For example, selecting the leftmost item in each given conflict is $\Omega(n)$-competitive as demonstrated by the instance in Figure 2. The same goes for the sequential strategy of picking the leftmost item among the items that were not eliminated in previous conflicts. (We assume that the top conflict is the first to arrive.)

2.2 Characterizing Optimal Solutions

The offline version of SIC can be reduced to maximum independent set in proper intervals graphs, which is solvable in polynomial time [5]. The reduction is as follows. First, remove all conflicts that are properly contained in other conflicts. It follows that there is a total order on the remaining conflicts, and therefore we may view each item as a node in a proper interval graph, where an interval is now a contiguous sequence of conflicts. Now finding an optimal schedule amounts to finding a maximum independent set in the above mentioned

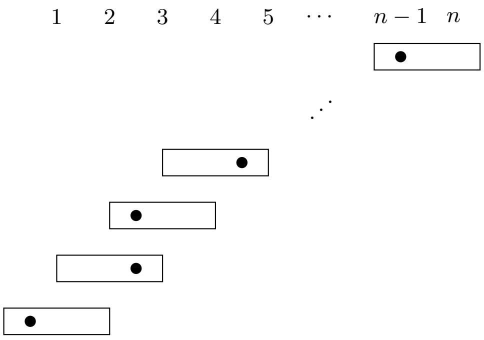


Figure 2 *An instance of* SIC *with n items and $n-1$ conflicts (all of size 2, i.e., $\sigma = 2$). If one selects the leftmost item in each given conflict, only one item (number 1) survives all conflicts, while by always picking the odd-numbered items (represented by dots in the figure), one gets an optimal solution of size $\lceil n/2 \rceil$.*

interval graph. Similarly, one may find a maximum independent set in an interval graph by solving an offline SIC instance.

We give a more direct description below. First, we provide an upper bound on the optimal solution due to duality. Let $\text{OPT}(\mathcal{C})$ denote an optimal solution of SIC to instance $\mathcal{C}$, and let $U(\mathcal{C}) = \bigcup_{C \in \mathcal{C}} C$ denote the set of items involved in conflicts in $\mathcal{C}$.

▶ **Observation 1.** *For all $\mathcal{C}' \subseteq \mathcal{C}$: If $U(\mathcal{C}') = U(\mathcal{C})$, then $|\text{OPT}(\mathcal{C}) \cap U(\mathcal{C})| \leq |\mathcal{C}'|$.*

Observation 1 motivates a simple polynomial offline algorithm for SIC. Briefly, the idea is to scan the item set from left to right (the examples in Figure 1 may help the reader), initially selecting the leftmost item. The next element selected, following a selected element i_j, is then inductively the leftmost element among those that are not in conflicts that contain i_j, i.e., $i_{j+1} = \min\{i' : i' > i_j \text{ and } \forall C, |\{i_j, i'\} \cap C| \leq 1\}$. This forms a feasible solution, since for any consecutively chosen items i_j and i_{j+1}, there is no conflict containing both i_j and i_{j+1}. To prove that the selected elements constitute an optimal solution, let C'_j be the conflict that contains i_j and $i_{j+1} - 1$. Since $\bigcup_j C'_j = U(\mathcal{C})$, optimality follows from Observation 1.

3 Oblivious Algorithms

In this section we consider oblivious algorithms. Oblivious algorithms are attractive because they can be implemented in a distributed system. The main result of this section is an oblivious algorithm for unweighted contiguous SIC, whose competitive ratio is $2 \lg \sigma$. We also show that if the instance is not contiguous, then no oblivious algorithm can be competitive.

3.1 Oblivious Algorithm for Contiguous SIC

In this section we present a simple $2 \lg \sigma$-competitive algorithm for unweighted contiguous SIC. We note that the algorithm needs not know σ in advance.

The basic idea of the algorithm is to assign to each item a fixed priority, and the conflict resolution rule is to always prefer the item with the highest priority. Specifically, our algorithm, **Priority**, defines the priority of item i by

$$p(i) \overset{\text{def}}{=} \max \left\{ \ell \in \mathbb{Z} \mid i \text{ is divisible by } 2^{\ell} \right\} \ . \tag{1}$$

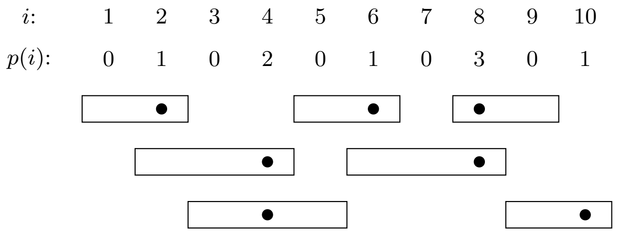


Figure 3 *Execution of Algorithm* **Priority** *on an instance with ten items and seven conflicts. The dots represent the items that were chosen by the algorithm. The computed solution is* $P = \{4, 8, 10\}$*, while the optimum is* $\{1, 3, 6, 9\}$.

For example, if i is odd, then $p(i) = 0$, and if $i = 2^\ell$ then $p(i) = \ell$.[1]

One nice consequence of this definition is the following observation:

▶ **Observation 2.** *If $i_\ell < i_r$ and $p(i_\ell) = p(i_r) = p$ for some p, then there exists $i_\ell < i < i_r$ such that $p(i) > p$.*

Observation 2 implies that any conflict contains exactly one item with maximum priority, and hence Algorithm **Priority** is well-defined: Upon arrival of conflict C, the algorithm delivers the unique item with highest priority (as defined by (1)) among the items in C. See Figure 3 for an example. Note that the algorithm makes decisions without knowing or even estimating σ, and that it is completely distributed: the identity of the winner of a conflict is independent of other conflicts.

Next, observe that even though σ, the size of the largest conflict, is unknown, we need only to concern ourselves with $\lg \sigma$ priorities.

▶ **Observation 3.** *Each conflict contains at most one item i with $p(i) \geq \lg \sigma$.*

Observation 3 implies that given conflicts whose length is bounded by σ, all priorities greater than or equal to $\lg \sigma$ are indistinguishable from the viewpoint of Algorithm **Priority**.

We now turn to prove that the competitive ratio of Algorithm **Priority** is at most $2 \lg \sigma$. We use the following concept.

▶ **Definition 4.** Let $\mathcal{C}$ be an instance and let A be an algorithm for SIC. A sequence of items $i_0, i_1, \ldots, i_m$ is an *elimination chain of length m* if for all $0 < j \leq m$ we have that item i_j eliminates item i_{j-1} when A runs on $\mathcal{C}$.

Note that an elimination chain of length m contains $m + 1$ items, but implies the existence of m conflict intervals. Elimination chains have the following property.

▶ **Lemma 5.** *Let $\mathcal{C}$ be an instance and let A be an algorithm for* SIC. *Suppose that $i_0, \ldots, i_m$ is an elimination chain for $\mathcal{C}$ under A. Then the interval $[i', i'']$ can be covered by m conflicts, where $i' = \min\{i_j \mid 0 \leq j \leq m\}$ and $i'' = \max\{i_j \mid 0 \leq j \leq m\}$.*

Proof. By Definition 4, for any $0 < j \leq m$ there exists a conflict $I_j \in \mathcal{I}$ such that $i_{j-1} \in I_j$ and $i_j \in I_j$. It follows, by induction on m, that $\bigcup_{j=1}^{m} I_j \supseteq [i', i'']$. ◀

[1] For an efficient implementation (in AC0), e.g. to use in routers, it suffices to extract the smallest bit set, using the bit-wise operations (i XOR $(i - 1)$) AND i.

Lemma 5 implies the following consequence. Say that an algorithm is *reasonable* if it delivers an item from each conflict C, i.e. it does not eliminate items without a reason.

▶ **Proposition 6.** *The competitive ratio of any reasonable algorithm for* SIC *is at most* $2m-1$, *where* m *is the length of the longest elimination chain of the algorithm.*

Proof. Fix an instance $\mathcal{C}$. Say that an item i' is *dominated* by an item i if there is an elimination chain that starts with i' and ends with i. For each item i, let $D(i)$ be the set of items dominated by i. Now, consider a reasonable algorithm A, and let P be the set of items delivered by A running on $\mathcal{C}$. Clearly, each item i was either delivered by A, i.e., $i \in P$, or eliminated, in which case $i \in D(i')$ for some $i' \in P$ (because A is reasonable). In addition, we have by Lemma 5 that $D(i')$ can be covered by the conflicts of two elimination chains: one starting from $\min(D(i'))$ and one that starts from $\max(D(i'))$. Both of them end in the same interval. We can therefore conclude that the set of all items can be covered by $(2m-1)|P|$ intervals. The result now follows from Observation 1. ◀

Using Proposition 6 and Observation 3 we can easily bound the competitive ratio of Algorithm **Priority**.

▶ **Theorem 7.** *The competitive ratio of Algorithm* ***Priority*** *is at most* $2\lg\sigma$.

Proof. By Observation 3 and the fact that under **Priority** an item i is eliminated by an item i' if and only $p(i) < p(i')$, we have that the length of any elimination chain under **Priority** is at most $\lg\sigma$. The theorem therefore follows directly from Proposition 6. ◀

In the full version of the paper we explain how to extend Algorithm **Priority** to the capacitated case, where the number of survivors from a conflict may be some parameter $b \geq 1$.

3.2 A Lower Bound for the Non-Contiguous Case

One may wonder if a similar result holds in the non-contiguous case. This turns out to be far from the case. We argue that no deterministic oblivious algorithm is competitive in the general (non-contiguous) case even if $\sigma = 2$.

▶ **Theorem 8.** *The competitive ratio of any deterministic oblivious algorithm for* SIC *is* $\Omega(n)$, *even for the unweighted case and for* $\sigma = 2$.

Proof. Fix a deterministic oblivious algorithm ALG. By definition of obliviousness, the decision of ALG for a given conflict C depends only on its items. Let n be a number and let $N = 2^n$. We 2-color the edges of an N-vertex clique K_N as follows. Edge (v_i, v_j), for $i < j$, is colored blue if ALG prefers v_i over v_j, and otherwise red. By Ramsey's theorem [4], K_N contains a monochromatic subgraph of $\log N = n$ vertices. If follows that there is either an increasing or a decreasing sequence of n items $i_1, \ldots, i_n$ such that ALG prefers i_ℓ over $i_{\ell-1}$, for any $\ell \in \{2, \ldots, n\}$. We introduce the conflicts $\{i_{\ell-1}, i_\ell\}$, for $\ell \in \{2, \ldots, n\}$. Then, only i_n will survive the execution of ALG, whereas $\{i_\ell : \ell \text{ is odd}\}$ is a feasible solution of size $n/2$. ◀

4 Sequential Algorithms

In this section we present an $O(\lg\sigma)$-competitive sequential algorithm for SIC, extending algorithm **Priority** to the weighted and non-contiguous case.

In describing our algorithm, we say that item i *fired* item i' if i was the first item to eliminate i'. In the remainder of this section we say that an item i_0 is *dominated* by item i_m if there is an elimination chain $i_0, i_1, \ldots, i_m$ such that i_j fires item i_{j-1}, for every j. Let $D(i)$ be the set of items that are dominated by i. Note that if i survives then $D(i) \neq \emptyset$, and in particular $i \in D(i)$. Observe that each item is dominated by exactly one surviving item, hence $D(i) \cap D(i') = \emptyset$ for each pair of surviving items i and i'.

We now describe the algorithm in the weighted case. Define the *weight class* of item i to be $c(i) = \lfloor \lg w(i) \rfloor$, the base-2 logarithm of the item weight rounded down to an integer. Our algorithm is called **Seq**, and it proceeds as follows.

With each item i, we associate two values $\text{left}(i)$ and $\text{right}(i)$ (initially both zero), referred to as the *left* and *right levels* of i, respectively. When an interval I arrives, the algorithm determines the highest weight class of active items in I. If there is only one active item of the highest weight class, it simply survives. Otherwise, let l and r be the leftmost and rightmost active items of the highest weight class. The algorithm compares $\text{left}(l)$, the left level of l, and $\text{right}(r)$, the right level of r. If $\text{left}(l) > \text{right}(r)$, then l survives and $\text{right}(l)$ is set to $\text{right}(r) + 1$; otherwise, r survives and $\text{left}(r)$ is set to $\text{left}(l) + 1$. (The algorithm arbitrarily favors r, in case of a tie.) Notice that $\text{left}(i)$ and $\text{right}(i)$ may increase and decrease during execution.

Fix some optimal solution OPT. The following upper bound is what motivates the numbering of the levels. Let m_i be the larger of the levels of i, namely $m_i = \max\{\text{left}(i), \text{right}(i)\}$. Also, let $n_i = \max_{i' \in D(i)} m_{i'}$ be the largest level of an item in $D(i)$.

▶ **Lemma 9.** $w(\text{OPT} \cap D(i)) = O(n_i \cdot w(i))$.

Proof. Let $l_1, \ldots, l_t$ be the sequence of items in $D(i)$ such that l_1 is the leftmost item in $D(i)$ and, inductively, l_{j+1} is the item that fired item l_j. Observe that the sequence extends monotonically from left to i, with $l_t = i$. According to the survival rule of the algorithm, the weight classes of the items are monotonically non-decreasing, and for a pair of items l_j and l_{j+1} in the same weight class, the left levels are strictly increasing, namely $\text{left}(l_{j+1}) > \text{left}(l_j)$. It follows that there are at most n_i items from the item set $\{l_1, \ldots, l_t\}$ in each weight class. Hence, the sum of the weights of the items $l_1, \ldots, l_t$ is bounded by

$$\sum_{j=1}^{t} w(l_j) < 2 \sum_{j=1}^{t} 2^{c(l_j)} \leq 2n_i \sum_{c \leq c(i)} 2^c < 2n_i \cdot 2^{c(i)+1} \leq 4n_i \cdot w(i) \ . \tag{2}$$

Let $D^-(i)$ ($D^+(i)$) be the subset of items in $D(i)$ to the left (right) of i, up to and including i. That is, $D^-(i) \cup D^+(i) = D(i)$ and $D^-(i) \cap D^+(i) = \{i\}$. Partition $D^-(i)$ into ranges $[l_{j+1}, l_j]$, for $j = 1, \ldots, t-1$. Observe that l_{j+1} must be in the largest weight class among the items in the range $[l_j, l_{j+1}]$, for all $j = 1, \ldots, t-1$. (Namely, if there was a item in $[l_j + 1, l_{j+1} - 1]$ belonging to a larger weight class, the largest such item could not have been eliminated without either l_j or l_{j+1} being also eliminated.) Since OPT can contain at most one item from each range $[l_j, l_{j+1}]$, it follows from (2) that

$$w(\text{OPT} \cap D^-(i)) < 2 \sum_{j=1}^{t} w(l_j) \leq 8n_i \cdot w(i) \ .$$

Applying the same arguments to $D^+(i)$ yields that $w(\text{OPT} \cap D(i)) \leq 16n_i \cdot w(i)$, implying the lemma. ◀

We now show that high levels imply very large intervals.

▶ **Lemma 10.** $n_i = O(\lg \sigma)$.

Proof. Let $\hat{D}(i')$ be the set of items dominated by i' that are from the same weight class, $c(i')$, as i'.

▶ Claim 1. $|\hat{D}(i')| \geq 2^{m_{i'}}$, for any item i' in $D(i)$.

Proof. The proof is by induction on $m_{i'}$. The base case $m_{i'} = 0$ is trivially true, since $i' \in \hat{D}(i')$. For the inductive step, suppose that $m_{i'} \geq 1$. Consider the most recent conflict I that i' survived and in which an item from class $c(i')$ was fired. Let l and r be the leftmost and rightmost active items in I, respectively, such that $c(l) = c(r) = c(i')$, when I was presented. Observe that $i' \in \{l, r\}$. By the inductive hypothesis, $|\hat{D}(l)| \geq 2^{m_l}$ and $|\hat{D}(r)| \geq 2^{m_r}$. If $m_{i'} = \max\{m_l, m_r\}$, then we are done. Suppose then that $m_{i'} = \max\{m_l, m_r\} + 1$, which happens only when $m_l = m_r$. Since $\hat{D}(l)$ and $\hat{D}(r)$ are disjoint, we have that

$$|\hat{D}(i')| \geq |\hat{D}(l)| + |\hat{D}(r)| \geq 2^{m_l} + 2^{m_r} = 2^{m_{i'}} .$$

and the claim follows. ◀

▶ Claim 2. $\hat{D}(i')$ is covered by at most $2m_{i'}$ intervals.

Proof. Let $l_1, l_2, \ldots, l_t$ be the sequence of items defined such that l_1 is the leftmost item in $\hat{D}(i')$ and, inductively, l_{j+1} is the item that fired l_j, for $j = 1, \ldots, t-1$. Also, let I_j be the interval presented upon which l_{j+1} fired l_j, for $j = 1, \ldots, t-1$. Clearly, $I_1, \ldots, I_{t-1}$ cover the items to the left of $l_t = i'$, up to and including i'. According to the survival rule of the algorithm, the left levels of the items are strictly increasing. It follows that $t \leq m_{i'} + 1$. By symmetry, $m_{i'}$ intervals also cover the items in $\hat{D}(i')$ to the right of i'. ◀

We resume with the proof of Lemma 10. Let $i' \in D(i)$ such that $n_i = m_{i'}$. By the two claims above, some interval covers at least $|\hat{D}(i')|/(2m_{i'}) \geq 2^{m_{i'}-1}/m_{i'}$ items. Hence, $\sigma \geq 2^{m_{i'}-1}/m_{i'}$, or $n_i = m_{i'} \leq \lg \sigma(1 + o(1))$. ◀

The following theorem is now immediate from Lemmas 9 and 10 when observing that the sets $\{D(i) : i \text{ survived}\}$ partition the set U of items.

▶ **Theorem 11.** *The competitive ratio of the oblivious algorithm* ***Seq*** *for the weighted and non-contiguous case is* $O(\lg \sigma)$.

5 A Lower Bound on the Competitive Ratio

In this section we show that the competitive ratio of any deterministic online algorithm for contiguous SIC is $\Omega(\lg \sigma)$. Our lower bound construction is sequential, namely the conflicts arrive one by one, and the algorithm knows the complete history when a new conflict arrives. Since any algorithm for oblivious SIC can be used in the sequential model, the lower bound holds for oblivious SIC as well.

Fix a deterministic online algorithm A. Based on the way A picks items to survive conflicts, we construct in an online fashion a sequence of conflicts along with an optimal scheduling denoted by OPT. To facilitate the description, define a conflict I to be *active* with respect to algorithm A if upon arrival, I contains a item that was not already eliminated by A in previous conflicts. W.l.o.g., we consider only algorithms that always deliver an item from an active interval.

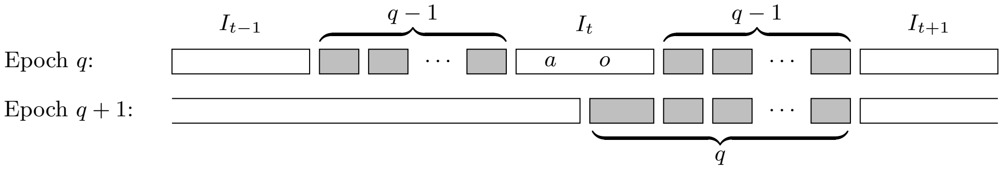


Figure 4 *Construction of epoch $q+1$: The case where $a < o$. Gray boxes represent positive intervals. I_t is split into two parts: the left part is combined with I_{t-1}, while the right part becomes a positive interval.*

In general, there can be conflicts that are active with respect to A, OPT, or both. We call a conflict interval *neutral* if it is active with respect to both A and OPT, and *positive* if it is active with respect to OPT only (there will be no "negative" intervals in our construction).

The conflict sequence consists of a sequence of *epochs* satisfying the following epoch invariant:

- In each epoch, all conflicts are disjoint and their union is $\{1, \ldots, n\}$.
- The set of items delivered by A and by OPT from epoch $q \geq 1$ are disjoint.
- In epoch q there are $q-1$ positive intervals between any two consecutive neutral intervals.

Note that the last property means that after epoch q, the optimal number of surviving items is q times larger than the number of items delivered by A.

We now describe the construction of epochs inductively. Assume that n is an even integer. The first epoch consists of $n/2$ intervals of size 2: for every $t \in [1, n/2]$, the tth interval is $[2t-1, 2t]$. Let A_1 be the set of items that are delivered by the algorithm after the first epoch. Clearly $|A_1| = n/2$. The optimal solution is the complement of A_1, namely $\text{OPT}_1 = \{1, \ldots, n\} \setminus A_1$. It is straightforward to verify that the epoch invariant holds for $q = 1$ (the last property follows from the fact that all intervals in epoch 1 are neutral).

The more interesting part is the inductive step. Let A_q and OPT_q be the set of active items with respect to A and OPT, respectively, immediately after epoch q. Assume that the invariant holds for epoch q. We construct epoch $q+1$ and OPT_{q+1} as follows. Number the neutral intervals of epoch q sequentially $I_1, I_2, \ldots$, starting from the leftmost neutral interval. This numbering skips the positive intervals between neutral intervals. Let $I_t = [\ell_t, r_t]$ be the tth neutral interval, where t is even. We break I_t into two parts as follows. Let a and o be the indices of items that are delivered from I_t by A_q and OPT_q, respectively in epoch q. We proceed by two cases. If $a < o$, then we introduce the conflict interval $[o, r_t]$ and extend I_{t-1} to the right up to $o-1$ (see Figure 4). Otherwise, if $a > o$, then we introduce the conflict interval $[\ell_t, o]$ and extend I_{t+1} to the left up to $o+1$ (see Figure 5). Notice that an odd neutral interval from epoch q can be either extended to the left, or to the right, or in both directions, or not extended at all. Finally, positive intervals from epoch q that are not covered by the above intervals are added to epoch $q+1$. Figure 6 illustrates a complete example.

It remains to determine OPT_{q+1}. Let I'_t be the extended version of an odd interval I_t from epoch q. OPT_{q+1} will deliver the active item from I_t. Since I_t does not intersect any even neutral interval, OPT_{q+1} may deliver an item in any part of an even neutral interval that was added to epoch $q+1$. Also, since I_t does not intersect any positive interval from epoch q, OPT_{q+1} may deliver an item in any positive interval that was not merged with an odd neutral interval from epoch q.

The first and second properties of the invariant are clearly satisfied by the construction.

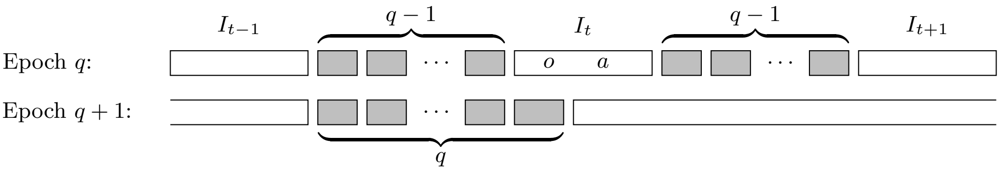


Figure 5 *Construction of epoch $q+1$: The case where $a > o$. Gray boxes represent positive intervals. I_t is split into two parts: the right part is combined with I_{t+1}, while the left part becomes a positive interval.*

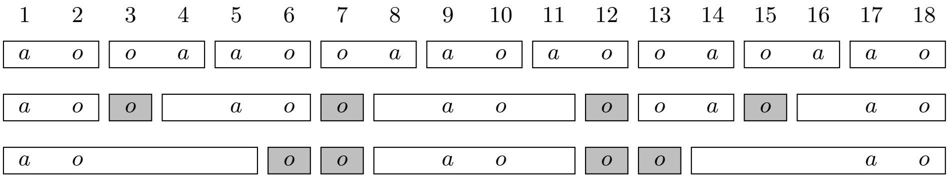


Figure 6 *The lower bound construction with $n = 18$ and three epochs. The gray boxes represent positive intervals.*

To see that the last property of the invariant holds, observe that any extended odd neutral interval remains neutral. We claim that there are q positive intervals between any consecutive neutral intervals. Let I_t be an even neutral interval that was split in the construction of epoch $q+1$. If $a < o$ the interval $[o, r_t]$ is positive because $[o, r_t] \cap A_q = \emptyset$. Moreover, all positive intervals between I_t and I_{t+1} remain as they were. Similarly, if $a > o$ the interval $[\ell_t, o]$ is positive, because $[\ell_t, o] \cap A_q = \emptyset$, and all positive intervals between I_t and I_{t-1} are left unchanged. Hence, there are q positive intervals between any two consecutive neutral intervals in epoch $q+1$.

The following lemma bounds the size of intervals.

▶ **Lemma 12.** *Let σ_q be the maximum interval size in epoch q. Then $\sigma_q \le 2 \cdot 5^{q-1}$.*

Proof. By induction on the number of epochs. In the base case (epoch 1), $\sigma_1 = 2$. For the inductive step, observe that an interval in the epoch $q+1$ may consist of (i) an odd neutral interval, (ii) parts of two even neutral intervals, (iii) $2(q-1)$ positive intervals. Since positive intervals that are created in epoch q' are of size smaller than $\sigma_{q'-1}$ and due to the inductive hypothesis, we have that

$$\sigma_{q+1} < \sigma_q + 2\sigma_q + 2\sum_{q'=1}^{q-1} \sigma_{q'} \le 3\sigma_q + 2\sum_{q'=1}^{q-1} \sigma_{q'} < 5\sigma_q \ ,$$

and the lemma follows. ◀

We can now prove the lower bound.

▶ **Theorem 13.** *The competitive ratio of any deterministic online algorithm for sequential* SIC *is $\Omega(\lg \sigma)$, even in the contiguous case.*

Proof. Let A be a deterministic algorithm. Construct instance $\mathcal{I}$ as described above. The epoch invariant implies that after q epochs, $|\text{OPT}(\mathcal{I})| \ge q(|A(\mathcal{I})| - 1)$. Hence, the competitive ratio of A is $\Omega(q)$. Let σ be a given parameter. By Lemma 12 we have that $\sigma_q \le 2 \cdot 5^{q-1}$, and therefore, setting $q = \lfloor \log_5(\sigma/2) \rfloor = \Omega(\lg \sigma)$ we have $\sigma_q \le \sigma$, and the result follows. ◀

6 Conclusion

In this paper we have introduced the problem of scheduling with interval conflicts and proved tight bounds on the competitive ratio of online algorithms to solve them. It would be interesting to consider other conflict topologies, and to understand to which degree randomness can help.

References

1 Baruch Awerbuch, Yair Bartal, Amos Fiat, and Adi Rosén. Competitive non-preemptive call control. In *Proc. 5th SODA*, pages 312–320, 1994.

2 Unnar Thor Bachmann, Magnús M. Halldórsson, and Hadas Shachnai. Online selection of intervals and t-intervals. In *Proc. 11th SWAT*, volume 6139 of *Lecture Notes in Computer Science*, pages 383–394, 2010.

3 Yuval Emek, Magnús M. Halldórsson, Yishay Mansour, Boaz Patt-Shamir, Jaikumar Radhakrishnan, and Dror Rawitz. Online set packing and competitive scheduling of multi-part tasks. In *Proc. 29th PODC*, pages 440–449, 2010.

4 Paul Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Math.*, 2:463–470, 1935.

5 M.C. Golumbic. *Algorithmic graph theory and perfect graphs.* Academic Press, New York, 1980.

6 Jerzy W. Jaromczyk, Andrzej Pezarski, and Maciej Slusarek. An optimal competitive algorithm for the minimal clique covering in circular arc graphs. In *Proc. 19th EWCG*, 2003.

7 David Karger, Cliff Stein, and Joel Wein. Scheduling algorithms. In Mikhail J. Atallah, editor, *Algorithms and Theory of Computation Handbook.* CRC Press, 1998.

8 Alexander Kesselman, Boaz Patt-Shamir, and Gabriel Scalosub. Competitive buffer management with packet dependencies. In *Proc. 23rd IPDPS*, pages 1–12, 2009.

9 R.J. Lipton and A. Tomkins. Online interval scheduling. In *Proc. 5th SODA*, pages 302–311, 1994.

10 Jiri Sgall. On-line scheduling. In *Online Algorithms*, volume 1442 of *Lecture Notes in Computer Science*, pages 196–231, 1996.

Analysis of multi-stage open shop processing systems*

Christian E.J. Eggermont[1], Alexander Schrijver[2], and Gerhard J. Woeginger[1]

1 **Department of Mathematics and Computer Science**
TU Eindhoven, The Netherlands
`c.e.j.eggermont@tue.nl; gwoegi@win.tue.nl`

2 **CWI and University of Amsterdam, Netherlands**
`lex@cwi.nl`

Abstract

We study algorithmic problems in multi-stage open shop processing systems that are centered around reachability and deadlock detection questions.

We characterize safe and unsafe system states. We show that it is easy to recognize system states that can be reached from the initial state (where the system is empty), but that in general it is hard to decide whether one given system state is reachable from another given system state. We show that the problem of identifying reachable deadlock states is hard in general open shop systems, but is easy in the special case where no job needs processing on more than two machines (by linear programming and matching theory), and in the special case where all machines have capacity one (by graph-theoretic arguments).

1998 ACM Subject Classification F.2.2

Keywords and phrases Scheduling; resource allocation; deadlock; computational complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.484

1 Introduction

We consider a *multi-stage open shop* processing system with n jobs $J_1, \ldots, J_n$ and m machines $M_1, \ldots, M_m$. Every job J_j $(j = 1, \ldots, n)$ requests processing on a certain subset $\mathcal{M}(J_j)$ of the machines; the ordering in which job J_j passes through the machines in $\mathcal{M}(J_j)$ is irrelevant and can be chosen arbitrarily by the scheduler. Every machine M_i $(i = 1, \ldots, m)$ has a corresponding *capacity* $\text{cap}(M_i)$, which means that at any moment in time it can simultaneously hold and process up to $\text{cap}(M_i)$ jobs. For more information on multi-stage scheduling systems, the reader is referred to the survey [6].

In this article, we are mainly interested in the performance of *real-time* multi-stage systems, where the processing time $p_{j,i}$ of job J_j on machine M_i is a priori unknown and hard to predict. The Central Control (the scheduling policy) of the system learns the processing time $p_{j,i}$ only when the processing of job J_j on machine M_i is completed. The various jobs move through the system in an unsynchronized fashion. Here is the standard behavior of a job in such a system:

[*] This research has been supported by the Netherlands Organisation for Scientific Research (NWO), grant 639.033.403; by DIAMANT (an NWO mathematics cluster); by the Future and Emerging Technologies unit of the European Community (IST priority), under contract no. FP6-021235-2 (project ARRIVAL); by BSIK grant 03018 (BRICKS: Basic Research in Informatics for Creating the Knowledge Society).

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 484–494
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

1. In the beginning the job is asleep and is waiting outside the system. For technical reasons, we assume that the job occupies an artificial machine M_0 of unbounded capacity.
2. After a finite amount of time the job wakes up, and starts looking for an available machine M on which it still needs processing. If the job detects such a machine M, it requests permission from the Central Control to move to machine M. If no such machine is available or if the Central Control denies permission, the job falls asleep again (and returns to the beginning of Step 2).
3. If the job receives permission to move, it releases its current machine and starts processing on the new machine M. While the job is being processed and while the job is asleep, it continuously occupies machine M (and blocks one of the cap(M) available places on M). When the processing of the job on machine M is completed and in case the job still needs processing on another machine, it returns to Step 2.
4. As soon as the processing of the job on all relevant machines is completed, the job informs the Central Control that it is leaving the system. We assume that the job then moves to an artificial final machine M_{m+1} (with unbounded capacity), and disappears.

The described system behavior typically occurs in robotic cells and flexible manufacturing systems. The high level goal of the Central Control is to arrive at the situation where all the jobs have been completed and left the system. Other goals are of course to reach a high system throughput, and to avoid unnecessary waiting times of the jobs. However special care has to be taken to prevent the system from reaching situations of the following type:

▶ **Example 1.** Consider an open shop system with three machines M_1, M_2, M_3 of capacity 1. There are three jobs that each require processing on all three machines. Suppose that the Central Control behaves as follows:

The first job requests permission to move to machine M_1. Permission granted.

The second job requests permission to move to machine M_2. Permission granted.

The third job requests permission to move to machine M_3. Permission granted.

Once the three jobs have completed their processing on theses machines, they keep blocking their machines and simultaneously keep waiting for the other machines to become idle. The processing never terminates. ◀

Example 1 illustrates a so-called *deadlock*, that is, a situation in which the system gets stuck and comes to a halt since no further processing is possible: Every job in the system is waiting for resources that are blocked by other jobs that are also waiting in the system. Resolving a deadlock is usually expensive (with respect to time, energy, and resources), and harmfully diminishes the system performance. In robotic cells resolving a deadlock typically requires human interaction. The scientific literature on deadlocks is vast, and touches many different areas like flexible manufacturing, automated production, operating systems, Petri nets, network routing, etc.

The literature distinguishes two basic types of system states (see for instance Coffman, Elphick & Shoshani [2], Gold [5], or Banaszak & Krogh [1]). A state is called *safe*, if there is at least one possible way of completing all jobs. A state is called *unsafe*, if every possible continuation eventually will get stuck in a deadlock. An example for a safe state is the initial situation where all jobs are outside the system (note that the jobs could move sequentially through the system and complete). Another example for a safe state is the final situation where all jobs have been completed. An example for an unsafe state are the deadlock states.

Summary of considered problems and derived results

In this article we study the behavior of safe and unsafe states in open shop scheduling systems. In particular, we investigate the computational complexity of the four algorithmic questions described in the following paragraphs. First, if one wants to have a smoothly running system, then it is essential to distinguish the safe from the unsafe system states:

PROBLEM: SAFE STATE RECOGNITION

INSTANCE: An open shop scheduling system. A system state s.

QUESTION: Is state s safe?

Section 3 provides a simple characterization of unsafe states, which leads to a (straightforward) polynomial time algorithm for telling safe states from unsafe states. Similar characterizations have already been given a decade ago in the work of Sulistyono & Lawley [9] and Xing, Lin & Hu [10]. Our new argument is extremely short and simple.

One of the most basic problems in analyzing a system consists in characterizing those system states that can be reached while the shop is running.

PROBLEM: REACHABLE STATE RECOGNITION

INSTANCE: An open shop scheduling system. A system state s.

QUESTION: Can the system reach state s when starting from the initial situation where all machines are still empty?

In Section 4 we derive a polynomial time algorithm for recognizing reachable system states. The main idea is to reverse the time axis, and to make the system run backward. Then reachable states in the original system translate into safe states in the reversed system, and the results from Section 3 can be applied.

Hence recognizing states that are reachable from the initial situation is easy. What about recognizing states that are reachable from some other given state?

PROBLEM: STATE-TO-STATE REACHABILITY

INSTANCE: An open shop scheduling system. Two system states s and t.

QUESTION: Can the system reach state t when starting from state s?

Surprisingly, there is a strong and sudden jump in the computational complexity of the reachability problem: Section 5 provides an NP-hardness proof for problem STATE-TO-STATE REACHABILITY.

Another fundamental question is whether an open shop system can ever fall into a deadlock. In case it cannot, then there are no reachable unsafe states and the CENTRAL CONTROL may permit all moves right away and without analyzing them; in other words the system is fool-proof and will run smoothly without supervision.

PROBLEM: REACHABLE DEADLOCK

INSTANCE: An open shop scheduling system.

QUESTION: Can the system ever reach a deadlock state when starting from the initial situation?

Section 6 proves problem REACHABLE DEADLOCK to be NP-hard, even for the highly restricted special case where the capacity of each machine is at most three and where each job requires processing on at most four machines. In Sections 7 and 8 we exhibit two special cases for which this problem is solvable in polynomial time: The special case where every job needs processing on at most two machines is settled by a linear programming formulation and techniques from matching theory. The special case where every machine has capacity one is solved by analyzing cycles in certain edge-colored graphs.

Because of the page limit, some of the proofs are missing and will only appear in the full version of the paper.

2 Basic definitions

A *state* of an open shop scheduling system is a snapshot describing a situation that might potentially occur while the system is running. A state s specifies for every job J_j

- the machine $M^s(J_j)$ on which this job is currently waiting or currently being processed,
- and the set $\mathcal{M}^s(J_j) \subseteq \mathcal{M}(J_j) - \{M^s(J_j)\}$ of machines on which the job still needs future processing.

The machines $M^s(J_j)$ implicitly determine

- the set $\mathcal{J}^s(M_i) \subseteq \{J_1, \ldots, J_n\}$ of jobs currently handled by machine M_i.

The *initial state* 0 is the state where all jobs are still waiting for their first processing; in other words in the initial state all jobs J_j satisfy $M^0(J_j) = M_0$ and $\mathcal{M}^0(J_j) = \mathcal{M}(J_j)$. The *final state* f is the state where all jobs have been completed; in other words in the final state all jobs J_j satisfy $M^f(J_j) = M_{m+1}$ and $\mathcal{M}^f(J_j) = \emptyset$.

A state t is called a *successor* of a state s, if it results from s by moving a single job J_j from its current machine $M^s(J_j)$ to some new machine in set $\mathcal{M}^s(J_j)$, or by moving a job J_j with $\mathcal{M}^s(J_j) = \emptyset$ from its current machine to M_{m+1}. In this case we will also say that the system *moves* from s to t. This successor relation is denoted $s \to t$. A state t is said to be *reachable* from state s, if there exists a finite sequence $s = s_0, s_1, \ldots, s_k = t$ of states (with $k \geq 0$) such that $s_{i-1} \to s_i$ holds for $i = 1, \ldots, k$. A state s is called *reachable*, if it is reachable from the initial state 0.

▶ **Lemma 2.** *Any reachable state s can be reached from the initial state through a sequence of at most $n + \sum_{i=1}^{n} |\mathcal{M}(J_j)|$ moves.* ◀

A state is called *safe*, if the final state f is reachable from it; otherwise the state is called *unsafe*. A state is a *deadlock*, if it has no successor states and if it is not the final state f.

3 Analysis of unsafe states

Unsafe states in open shop systems are fairly well-understood, and the literature contains several characterizations for them; see for instance Sulistyono & Lawley [9], Xing, Lin & Hu [10], and Lawley [7]. In this section we provide yet another analysis of unsafe states, which is shorter and (as we think) simpler than the previously published arguments.

A machine M is called *full* in state s, if it is handling exactly $\text{cap}(M)$ jobs. A non-empty subset $\mathcal{B}$ of the machines is called *blocking* for state s,

- if every machine in $\mathcal{B}$ is full, and
- if every job J_j that occupies some machine in $\mathcal{B}$ satisfies $\emptyset \neq \mathcal{M}^s(J_j) \subseteq \mathcal{B}$.

Here is a simple procedure that determines whether a given machine M_i is part of a blocking set in state s: Let $\mathcal{B}_0 = \{M_i\}$. For $k \geq 1$ let $\mathcal{J}_k$ be the union of all job sets $\mathcal{J}^s(M)$ with $M \in \mathcal{B}_{k-1}$, and let $\mathcal{B}_k$ be the union of all machine sets $\mathcal{M}^s(J)$ with $J \in \mathcal{J}_k$. Clearly $\mathcal{B}_0 \subseteq \mathcal{B}_1 \subseteq \cdots \subseteq \mathcal{B}_{m-1} = \mathcal{B}_m$. Furthermore machine M_i belongs to a blocking set, if and only if $\mathcal{B}_m$ is a blocking set, if and only if all machines in $\mathcal{B}_m$ are full. In case $\mathcal{B}_m$ is a blocking set, we denote it by $\mathcal{B}^s_{\min}(M_i)$ and call it the *canonical* blocking set for machine M_i in state s. The canonical blocking set is the smallest blocking set containing M_i:

▶ **Lemma 3.** *If machine M_i belongs to a blocking set $\mathcal{B}$ in state s, then $\mathcal{B}^s_{\min}(M_i) \subseteq \mathcal{B}$.* ◀

The machines in a blocking set $\mathcal{B}$ all operate at full capacity on jobs that in the future only want to move to other machines in $\mathcal{B}$. Since these jobs are permanently blocked from moving, the state s must eventually lead to a deadlock and hence is unsafe. The following theorem shows that actually *every* deadlock is caused by such blocking sets.

▶ **Theorem 4.** *A state s is unsafe if and only if it has a blocking set of machines.*

Proof. The if-statement is obvious. For the only-if-statement, we classify the unsafe states with respect to their distances to deadlock states. The set $\mathcal{U}_0$ contains the deadlock states. For $d \geq 1$, set $\mathcal{U}_d$ contains all states whose successor states are all contained in $\mathcal{U}_{d-1}$. Note that $\mathcal{U}_{d-1} \subseteq \mathcal{U}_d$, and note that every unsafe state occurs in some $\mathcal{U}_d$. We prove by induction on d that every state in $\mathcal{U}_d$ has a blocking set of machines. For $d = 0$ this is trivial.

In the inductive step, assume for the sake of contradiction that some state $s \in \mathcal{U}_d$ is unsafe but does not contain any blocking set. Since every move from s leads to a state in $\mathcal{U}_{d-1}$, all successor states of s must contain blocking sets. Whenever in state s some job J moves to some (non-full) machine M, this machine M must become full and must then be part of any blocking set. Among all possible moves, consider a move that yields a state t with a newly full machine M for which the canonical blocking set $\mathcal{B}^t_{\min}(M)$ is of the smallest possible cardinality.

Note that in state t there exist a machine $M' \in \mathcal{B}^t_{\min}(M)$ and a job $J' \in \mathcal{J}^t(M')$ with $M \in \mathcal{M}^t(J')$; otherwise $\mathcal{B}^t_{\min}(M) - \{M\}$ would be a blocking set for state s. Now consider the successor state u of s that results by moving job J' from machine M to M'. Since $\mathcal{M}^u(J') \subseteq \mathcal{B}^t_{\min}(M)$, a simple inductive argument shows that $\mathcal{B}^u_{\min}(M) \subseteq \mathcal{B}^t_{\min}(M)$. Since job J' has just jumped away from M', this machine cannot be full in state u, and hence $M' \in \mathcal{B}^t_{\min}(M) - \mathcal{B}^u_{\min}(M)$. Consequently the canonical blocking set $\mathcal{B}^u_{\min}(M)$ has smaller cardinality than $\mathcal{B}^t_{\min}(M)$. This contradiction completes the proof. ◀

▶ **Lemma 5.** *For a given state s, it can be decided in polynomial time whether s has a blocking set of machines. Consequently, problem* Safe State Recognition *can be decided in polynomial time.*

Proof. Create an auxiliary digraph that corresponds to state s: the vertices are the machines $M_1, \ldots, M_m$. Whenever some job J_j occupies a machine M_i, the digraph contains an arc from M_i to every machine in $\mathcal{M}^s(J_j)$. Obviously state s has a blocking set of machines if and only if the auxiliary digraph contains a strongly connected component with the following two properties: (i) All vertices in the component are full. (ii) There are no arcs leaving the component. Since the strongly connected components of a digraph can easily be determined and analyzed in linear time (see for instance [3]), the desired statement follows. ◀

4 Analysis of reachable states

In this section we discuss the behavior of reachable system states. We say that a state t is *subset-reachable* from state s, if every job J_j satisfies one of the following three conditions:

- $M^t(J_j) = M^s(J_j)$ and $\mathcal{M}^t(J_j) = \mathcal{M}^s(J_j)$, or
- $M^t(J_j) \in \mathcal{M}^s(J_j)$ and $\mathcal{M}^t(J_j) \subseteq \mathcal{M}^s(J_j) - \{M^t(J_j)\}$, or
- $M^t(J_j) = M_{m+1}$ and $\mathcal{M}^t(J_j) = \emptyset$.

Clearly whenever a state t is reachable from some state s, then t is also subset-reachable from s. The following example demonstrates that the reverse implication is not necessarily true. This example also indicates that the algorithmic problem REACHABLE STATE RECOGNITION (as formulated in the introduction) is not completely straightforward.

▶ **Example 6.** Consider an open shop system with two machines M_1, M_2 of capacity 1 and two jobs J_1, J_2 with $\mathcal{M}(J_1) = \mathcal{M}(J_2) = \{M_1, M_2\}$. Consider the state s where J_1 is being processed on M_1 and J_2 is being processed on M_2, and where $\mathcal{M}^s(J_1) = \mathcal{M}^s(J_2) = \emptyset$. It can be seen that s is subset-reachable from the initial state 0, whereas s is not reachable from 0. ◀

Our next goal is to derive a polynomial time algorithm for recognizing reachable system states. Consider an open shop scheduling system and a fixed system state s. Without loss of generality we assume that s is subset-reachable from the initial state. We define a new (artificial) state t where $M^t(J_j) := M^s(J_j)$ and $\mathcal{M}^t(J_j) := \mathcal{M}(J_j) - \mathcal{M}^s(J_j) - \{M^s(J_j)\}$ for all jobs J_j. Note that in both states s and t every job is sitting on the very same machine, but the work that has already been performed in state s is exactly the work that still needs to be done in state t.

▶ **Lemma 7.** *State s is reachable if and only if state t is safe.*

Proof. First assume that s is reachable, and let $0 = s_0 \to s_1 \to \cdots \to s_k = s$ denote a corresponding witness sequence of moves. Define a new sequence $t = t_k \to t_{k-1} \to \cdots \to t_0 = f$ of moves: Whenever the move $s_\ell \to s_{\ell+1}$ $(0 \le \ell \le k-1)$ results from moving job J_j from machine M_a to machine M_b, then the move $t_{\ell+1} \to t_\ell$ results from moving job J_j from machine M_b to machine M_a. (Note that the artificial machines M_0 and M_{m+1} switch their roles.) Hence t is safe. A symmetric argument shows that if t is safe then s is reachable. ◀

Hence deciding reachability is algorithmically equivalent to deciding safeness. Together with Lemma 5 this yields the following theorem.

▶ **Theorem 8.** REACHABLE STATE RECOGNITION *can be decided in polynomial time.* ◀

The following lemma states a simple sufficient condition that makes a state reachable.

▶ **Lemma 9.** *Let s be a state, and let $\mathcal{K}$ be a subset of machines such that every job that still needs further processing in s satisfies $M^s(J_j) \in \mathcal{K}$ and*

$$\mathcal{M}^s(J_j) \cup \{M^s(J_j)\} \;=\; \mathcal{K} \cap \mathcal{M}(J_j).$$

Then s is a reachable system state.

Proof. By renaming the jobs we assume that the jobs J_j with $1 \le j \le k$ have $M^s(J_j) = M_{m+1}$ and the jobs J_j with $k+1 \le j \le n$ have $M^s(J_j) \in \mathcal{K}$. We handle the jobs one by one in their natural order: every job moves through all machines in $\mathcal{M}(J_j) - \mathcal{M}^s(J_j)$, and ends up on machine $M^s(J_j)$. Then the next job is handled. ◀

5 Analysis of state-to-state reachability

We establish NP-hardness of STATE-TO-STATE REACHABILITY by means of a reduction from the following satisfiability problem; see Garey & Johnson [4].

Problem: Three-Satisfiability

Input: A set $X = \{x_1, \ldots, x_n\}$ of n logical variables; a set $C = \{c_1, \ldots, c_m\}$ of m clauses over X that each contain three literals.

Question: Is there a truth assignment for X that satisfies all clauses in C?

We start from an instance of Three-Satisfiability, and construct a corresponding instance of State-to-State Reachability for it. Throughout we will use ℓ_i to denote the unnegated literal x_i or the negated literal $\overline{x_i}$ for some fixed variable $x_i \in X$, and we will use ℓ to denote a generic literal over X. Altogether there are $5n + m$ machines:

- For every literal ℓ_i, there are three corresponding machines $S(\ell_i)$, $T(\ell_i)$, and $U(\ell_i)$. Machine $U(\ell_i)$ has capacity 2, whereas machines $S(\ell_i)$ and $T(\ell_i)$ have capacity 1. For every variable $x_i \in X$ the two machines $U(x_i)$ and $U(\overline{x_i})$ coincide, and the corresponding machine will sometimes simply be called $U(i)$.
- For every clause $c_j \in C$, there is a corresponding machine $V(c_j)$ with capacity 3.

Furthermore the scheduling instance contains $4n$ jobs that correspond to literals and $6m$ jobs that correspond to clauses. For every literal ℓ_i there are two corresponding jobs:

- Job $J(\ell_i)$ is sitting on machine $S(\ell_i)$ in state s. In state t it has moved to machine $U(\ell_i)$ without visiting other machines inbetween.
- Job $J'(\ell_i)$ is still waiting outside the system in state s, and has already left the system in state t. Inbetween the job visits machines $S(\ell_i)$, $T(\ell_i)$, $U(\ell_i)$ in arbitrary order.

Consider a clause c_j that consists of three literals ℓ_a, ℓ_b, ℓ_c. Then the following six jobs correspond to clause c_j:

- For $\ell \in \{\ell_a, \ell_b, \ell_c\}$ there is a job $K(c_j, \ell)$ that in state s sits on machine $V(c_j)$, then moves through machines $S(\ell)$ and $T(\ell)$ in arbitrary order, and finally has left the system in state t. Note that in state s these three jobs block machine $V(c_j)$ to full capacity.
- For $\ell \in \{\ell_a, \ell_b, \ell_c\}$ there is another job $K'(c_j, \ell)$ that waits outside the system in state s, then moves through machines $U(\ell)$ and $V(c_j)$ in arbitrary order, and finally has left the system in state t.

In the full version of the paper, we will show that in the constructed scheduling instance state t is reachable from state s if and only if the Three-Satisfiability instance has a satisfying truth assignment. This then implies the following theorem.

▶ **Theorem 10.** State-to-State Reachability *is NP-complete.* ◀

6 Analysis of reachable deadlocks

In this section we show that Reachable Deadlock is an NP-hard problem. Our reduction is from the following variant of the Three-Dimensional Matching problem; see Garey & Johnson [4, p.221].

Problem: Three-Dimensional Matching

Instance: An integer n. Three pairwise disjoint sets $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$, and $C = \{c_1, \ldots, c_n\}$. A set $T \subseteq A \times B \times C$ of triples, such that every element occurs in at most three triples in T.

Question: Does there exist a subset $T' \subseteq T$ of n triples, such that every element in $A \cup B \cup C$ occurs in exactly one triple in T'?

We start from an arbitrary instance of Three-Dimensional Matching, and construct the following corresponding instance of Reachable Deadlock for it. There are two types of machines. Note that every machine has capacity at most three.

- There are $n+2$ so-called *structure machines* $S_0, \dots, S_{n+1}$, each of capacity 1.
- For every triple $t \in T$, there is a corresponding *triple machine* T_t with capacity 3.

Furthermore there are $4n+2$ jobs.

- For every element $a_i \in A$ there are two corresponding *A-element jobs* $J^+(a_i)$ and $J^-(a_i)$. Job $J^+(a_i)$ requires processing on structure machine S_i, and on every triple machine T_t with $a_i \in t$. Job $J^-(a_i)$ requires processing on structure machine S_{i-1}, and on every triple machine T_t with $a_i \in t$.
- For every element $b_i \in B$ there is a corresponding *B-element job* $J(b_i)$ that requires processing on structure machine S_{n+1}, and on every triple machine T_t with $b_i \in t$.
- For every element $c_i \in C$ there is a corresponding *C-element job* $J(c_i)$ that requires processing on structure machine S_{n+1}, and on every triple machine T_t with $c_i \in t$.
- Finally there is a dummy job D_0 that needs processing on S_0 and S_{n+1}, and another dummy job D_{n+1} that needs processing on S_n and S_{n+1}.

Since every element of $A \cup B \cup C$ occurs in at most three triples, we note that each job requires processing on at most four machines. For the ease of later reference, we also list for every machine the jobs that need processing on that machine.

- A triple machine T_t with $t = (a_i, b_j, c_k)$ handles the four jobs $J^+(a_i)$, $J^-(a_i)$, $J(b_j)$, and $J(c_k)$.
- Structure machine S_i with $1 \le i \le n-1$ handles the jobs $J^+(a_i)$ and $J^-(a_{i+1})$.
  Structure machine S_0 handles the two jobs $J^-(a_1)$ and D_0.
  Structure machine S_n handles the two jobs $J^+(a_n)$ and D_{n+1}.
  Structure machine S_{n+1} handles $2n+2$ jobs: D_0, D_{n+1}, all B-element jobs, and all C-element jobs.

The following theorem contains the main result of this section.

▶ **Theorem 11.** Reachable Deadlock *is NP-complete, even if the capacity of each machine is at most three, and if each job requires processing on at most four machines.* ◀

Indeed, Lemma 2 yields an NP-certificate for problem Reachable Deadlock. The hardness argument proves that the constructed scheduling instance has a reachable deadlock if and only if the Three-Dimensional Matching instance has answer YES. All details are provided in the full version of the paper.

7 Reachable deadlocks if jobs require two machines

Throughout this section we only consider open shop systems where $|\mathcal{M}(J)| = 2$ holds for all jobs J. We introduce for every job J and for every machine $M \in \mathcal{M}(J)$ a corresponding real variable $x(J, M)$, and for every machine M a corresponding real variable $y(M)$. Our analysis is centered around the following linear program (LP):

$$\begin{array}{lll} \min & \sum_M \max\{y(M),\ \mathrm{cap}(M)\} & \\ \text{s.t.} & \sum_{J: M \in \mathcal{M}(J)} x(J, M) \ = \ y(M) & \text{for all machines } M \\ & \sum_{M \in \mathcal{M}(J)} x(J, M) \ = \ 1 & \text{for all jobs } J \\ & x(J, M) \ \ge \ 0 & \text{for all } J \text{ and } M \in \mathcal{M}(J) \end{array}$$

Although this linear program is totally unimodular, we will mainly deal with its fractional solutions.

▶ **Lemma 12.** *One can compute in polynomial time an optimal solution for the linear program (LP) that additionally satisfies the following property (*) for every job J with $\mathcal{M}(J) = \{M_a, M_b\}$: If $y(M_a) \geq \mathrm{cap}(M_a)$ and $x(J, M_a) > 0$, then $y(M_b) \geq \mathrm{cap}(M_b)$.*

Proof. We determine in polynomial time an optimal solution of (LP). Then we perform a polynomial number of post-processing steps on this optimal solution, as long as there exists a job violating property (*). In this case $y(M_a) \geq \mathrm{cap}(M_a)$, $x(J, M_a) > 0$, and $y(M_b) < \mathrm{cap}(M_b)$.

The post-processing step decreases the values $x(J, M_a)$ and $y(M_a)$ by some $\varepsilon > 0$, and simultaneously increases $x(J, M_b)$ and $y(M_b)$ by the same ε. By picking ε smaller than the minimum of $\mathrm{cap}(M_b) - y(M_b)$ and $x(J, M_a)$ this will yield another feasible solution for (LP). What happens to the objective value? If $y(M_a) > \mathrm{cap}(M_a)$ at the beginning of the step, then the step would decrease the objective value, which contradicts optimality. If $y(M_a) = \mathrm{cap}(M_a)$ at the beginning of the step, then the step leaves the objective value unchanged, and yields another optimal solution with $y(M_a) < \mathrm{cap}(M_a)$ and $y(M_b) < \mathrm{cap}(M_b)$.

To summarize, every post-processing step decreases the number of machines M with $y(M) = \mathrm{cap}(M)$. Hence the entire procedure terminates after at most m steps. ◀

Let $x^*(J, M)$ and $y^*(M)$ denote an optimal solution of (LP) that satisfies the property (*) in Lemma 12. Let $\mathcal{M}^*$ be the set of machines M with $y^*(M) \geq \mathrm{cap}(M)$.

▶ **Lemma 13.** *The open shop system has a reachable deadlock, if and only if $\mathcal{M}^* \neq \emptyset$.*

Proof. (Only if). Consider a reachable deadlock state, let $\mathcal{B}'$ be the corresponding blocking set of machines, and let $\mathcal{J}'$ be the set of jobs waiting on these machines. Every job $J \in \mathcal{J}'$ is sitting on some machine in $\mathcal{B}'$, and is waiting for some other machine in $\mathcal{B}'$. Since $|\mathcal{M}(J)| = 2$, this implies $\mathcal{M}(J) \subseteq \mathcal{B}'$ for every job $J \in \mathcal{J}'$. Then

$$\sum_{M \in \mathcal{B}'} y^*(M) \;\geq\; \sum_{J \in \mathcal{J}'} \sum_{M \in \mathcal{M}(J)} x^*(J, M) \;=\; |\mathcal{J}'|.$$

Since furthermore $|\mathcal{J}'| = \sum_{M \in \mathcal{B}'} \mathrm{cap}(M)$, we conclude $y^*(M) \geq \mathrm{cap}(M)$ for at least one machine $M \in \mathcal{B}'$.

(If). Let $\mathcal{J}^*$ be the set of jobs with $x^*(J, M) > 0$ for some $M \in \mathcal{M}^*$. Property (*) in Lemma 12 now yields the following for every job J: If $J \in \mathcal{J}^*$, then $\mathcal{M}(J) \subseteq \mathcal{M}^*$. Construct a bipartite graph G between the jobs in $\mathcal{J}^*$ and the machines in $\mathcal{M}^*$, with an edge between J and M if and only if $M \in \mathcal{M}(J)$. For any subset $\mathcal{M}' \subseteq \mathcal{M}^*$, the number of job neighbors in this bipartite graph is at least $\sum_{M \in \mathcal{M}'} y^*(M) \geq \sum_{M \in \mathcal{M}'} \mathrm{cap}(M)$. A variant of Hall's theorem from matching theory [8] now yields that there exists an assignment of some jobs from $\mathcal{J}^*$ to machines in $\mathcal{M}^*$ such that every $M \in \mathcal{M}^*$ receives $\mathrm{cap}(M)$ pairwise distinct jobs.

To reach a deadlock, we first send all non-assigned jobs one by one through the system. They are completed and disappear. Then the assigned jobs enter the system, each moving straightly to the machine to which it has been assigned. Then the system falls into a deadlock with blocking set $\mathcal{M}^*$: All machines in $\mathcal{M}^*$ are full, and all jobs are only waiting for machines in $\mathcal{M}^*$. ◀

Since jobs J with $|\mathcal{M}(J)| = 1$ are harmless and may be disregarded with respect to deadlocks, we arrive at the following theorem.

▶ **Theorem 14.** *For open shop systems where each job requires processing on at most two machines,* Reachable Deadlock *can be solved in polynomial time.* ◀

The following example illustrates that the above LP-based approach cannot be carried over to the case where every job requires processing on three machines (since the only-if part of Lemma 13 breaks down).

▶ **Example 15.** Consider a system with two jobs and four machines of unit capacity. Job J_1 needs processing on M_1, M_2, M_3, and job J_2 needs processing on M_1, M_2, M_4. A (reachable) deadlock results if J_1 enters the system on M_3 and then moves to M_1, whereas J_2 simultaneously enters the system on M_4 and then moves to M_2.

We consider a feasible solution with $x(J, M) \equiv 1/3$ for every J and every $M \in \mathcal{M}(J)$, and $y(M_1) = y(M_2) = 2/3$ and $y(M_3) = y(M_4) = 1/3$. The objective value is 4, and hence this is an optimal solution. The post-processing leaves the solution untouched, and the resulting set $\mathcal{M}^*$ is empty. ◀

8 Reachable deadlocks if machines have unit capacity

Throughout this section we only consider open shop systems with $\text{cap}(M_i) \equiv 1$. For each such system we define a corresponding undirected edge-colored multi-graph $G = (V, E)$: The vertices are the machines $M_1, \dots, M_m$. Every job J_j induces a clique of edges on the vertex set $\mathcal{M}(J_j)$, and all these edges receive color c_j. Intuitively, if two machines are connected by an edge e of color c_j, then job J_j may move between these machines along edge e.

▶ **Lemma 16.** *For an open shop system with unit machine capacities and its corresponding edge-colored multi-graph the following two statements are equivalent.*
(i) The multi-graph contains a simple cycle whose edges have pairwise distinct colors.
(ii) The system can reach a deadlock.

Proof. Assume that (i) holds, and consider a simple cycle C whose edges have pairwise distinct colors. By renaming jobs and machines we may assume that the vertices in C are the machines $M_1, \dots, M_k$, and that the edges in C are $[M_j, M_{j+1}]$ with color c_j for $1 \le j \le k-1$, and $[M_k, M_1]$ with colors c_k. Consider the following processing order of the jobs:

- In the first phase, the jobs J_j with $k + 1 \le j \le n$ are processed one by one: Job J_{j+1} only enters the system after job J_j has completed all its processing and has already left the system. At the end of this phase we are left with the jobs $J_1, \dots, J_k$.
- In the second phase, the jobs $J_1, \dots, J_k$ are handled one by one. When job J_j is handled, first all operations of J_j on machines M_i with $i \ge k+1$ are processed. Then job J_j moves to machine M_j, and stays there till the end of the second phase. Then the next job is handled.

At the end of the second phase, for $1 \le i \le k$ job J_i is blocking machine M_i, and waiting for future processing on some other machine in cycle C. The system has fallen into a deadlock, and hence (i) implies (ii).

Next assume that (ii) holds, and consider a deadlock state. For every waiting job J_j in the deadlock, let M'_j be the machine on which J_j is currently waiting and let M''_j denote one of the machines for which the job is waiting. Consider the sub-graph of G that for every waiting job J_j contains the vertex M'_j together with an edge $[M'_j, M''_j]$ of color c_j. This sub-graph has as many vertices as edges, and hence must contain a simple cycle; hence (ii) implies (i). ◀

▶ **Lemma 17.** *For the edge-colored multi-graph $G = (V, E)$ corresponding to some open shop system with unit machine capacities, the following three statements are equivalent.*

(i) The multi-graph contains a simple cycle whose edges have pairwise distinct colors.
(ii) The multi-graph contains a 2-vertex-connected component that spans edges of at least two different colors.
(iii) The multi-graph contains a simple cycle whose edges have at least two different colors.

Proof. We show that (i) implies (ii) implies (iii) implies (i). The implication from (i) to (ii) is straightforward.

Assume that (ii) holds, and consider a vertex v in such a 2-vertex-connected component that is incident to two edges with two distinct colors. These two edges can be connected to a simple cycle, and we get (iii).

Assume (iii), and consider the shortest cycle C whose edges have at least two different colors. If two edges $[u,u']$ and $[v,v']$ on C have the same color c_j, then the vertices u,u',v,v' are all in the machine set $\mathcal{M}(J_j)$ of job J_j. Hence they span a clique in color c_j, and some edges in this clique can be used to construct a shorter cycle with edges of at least two different colors. This contradiction shows that (iii) implies (i). ◀

Lemmas 16 and 17 together yield that an open shop system can fall into a deadlock state if and only if the corresponding multi-graph contains a 2-vertex-connected component that spans edges of at least two different colors. Since the 2-vertex-connected components of a graph can easily be determined and analyzed in linear time (see for instance [3]), we arrive at the following theorem.

▶ **Theorem 18.** *For open shop systems with unit machine capacities, problem* Reachable Deadlock *can be solved in polynomial time.* ◀

References

1 Z.A. Banaszak and B.H. Krogh (1990). Deadlock avoidance in flexible manufacturing systems with concurrently competing process flows. *IEEE Transactions on Robotics and Automation 6*, 724–734.

2 E.G. Coffman, M.J. Elphick, and A. Shoshani (1971). System Deadlocks. *ACM Computing Surveys 3*, 67–78.

3 T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein (2001). *Introduction to Algorithms.* MIT Press.

4 M.R. Garey and D.S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, San Francisco.

5 M. Gold (1978). Deadlock prediction: Easy and difficult cases. *SIAM Journal on Computing 7*, 320–336.

6 E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (1993). Sequencing and scheduling: Algorithms and complexity. In: *Handbooks in Operations Research and Management Science, Vol. 4*, North Holland, 445–522.

7 M. Lawley (1999). Deadlock avoidance for production systems with flexible routing. *IEEE Transactions on Robotics and Automation 15*, 497–510.

8 L. Lovász and M.D. Plummer (1986). Matching Theory. *Annals of Discrete Mathematics 29*, North-Holland.

9 W. Sulistyono and M. Lawley (2001). Deadlock avoidance for manufacturing systems with partially ordered process plans. *IEEE Transactions on Robotics and Automation 17*, 819–832.

10 K. Xing, F. Lin, and B. Hu (2001). An optimal deadlock avoidance policy for manufacturing system with flexible operation sequence and flexible routing. *Proceedings of the 2001 IEEE International Conference on Robotics and Automation (ICRA'2001)*, 3565–3570.

Graphs Encoded by Regular Expressions

Stefan Gulan

Universität Trier, FB IV — Informatik
gulan@uni-trier.de

Abstract

In the conversion of finite automata to regular expressions, an exponential blowup in size can generally not be avoided. This is due to graph-structural properties of automata which cannot be directly encoded by regular expressions and cause the blowup combinatorially. In order to identify these structures, we generalize the class of arc-series-parallel digraphs to the acyclic case. The resulting digraphs are shown to be reversibly encoded by linear-sized regular expressions. We further derive a characterization of our new class by a finite set of forbidden minors and argue that these minors constitute the primitives causing the blowup in the conversion from automata to expressions.

1998 ACM Subject Classification F.4.3: Formal Languages, G.2.2: Graph Theory

Keywords and phrases Digraphs, Regular Expressions, Finite Automata, Forbidden Minors

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.495

1 Motivation

A fundamental result in the theory of regular languages is the equivalent descriptive power of regular expressions and finite automata, as originally shown by Kleene [9]. While regular expressions come natural to humans as a means of denoting such languages, automata are the objects of choice on the machine level. Consequently, converting between these two representations is of great practical importance. There are several linear-time algorithms to transform regular expressions into automata with size linear in that of the input, a detailed overview is given by Watson [16]. We shall focus on the converse construction which is considerably more troubling.

In particular, Ehrenfeucht & Zeiger [3] give a class of automata for which the size of any equivalent expression is exponential in that of a given automaton. These automata are defined over an alphabet which grows with automaton size, which led Ellul et al. to ask whether a similar blowup in expression size can be shown for automata over a fixed alphabet [4]. An affirmative answer was given by Gruber & Holzer [5] for binary alphabets already. This mostly rules out alphabet size as a factor contributing to the exponential blowup, the modifier 'mostly' giving credit to the fact that automata over unary alphabets can be converted to expressions of quadratic size via Chrobak normal form [1, 4].

Observe that a finite automaton is merely a digraph with edge labels, accepting the language which consists of all sequences of labels met on a directed walk from an initial to a final state. Informally, the increase of expression- over automaton-size results from automata being combinatorial objects, whereas expressions are terms, i.e., linear entities, that must resort to repeated subterms in order to convey information encoded in the graph-structure of an automaton. This was observed quite early by McNaughton [11], who remarks that "although every regular expression can be transformed into a graph that has the same structure, the converse is not true". The present work aims to identify the graphs that cannot be transformed into expressions that have the same structure.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 495–506
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

For automata whose underlying graphs are arc-series-parallel, Moreira & Reis [12] recently gave an efficient conversion to expressions with size linear in that of the input. These graphs are characterized by absence of a single minor-like substructure in acyclic graphs, as was shown by Valdes et al. [15]. Korenblit & Levit [10] conjectured that this substructure already causes a quadratic blowup in the size of expressions constructed from acyclic automata.

However, Moreira & Reis's method is inherently confined to automata that accept finite languages only. In order to accept an infinite language, an automaton needs to contain cycles, which evades the class of arc-series-parallel digraphs. While separately dealing with series-parallel 'parts' of arbitrary automata has been suggested for conversion-heuristics [6], no strict graph-theoretic analysis has been conducted for the general case as yet.

This motivates our generalization of arc-series-parallel digraphs to the non-acyclic case in Sec. 3, yielding a class which is still efficiently recognizable. In Sec. 4 we show that such graphs can be reversibly encoded by regular expressions and that every regular expression encodes a graph of this class. Encoding and decoding is done in an automata-theoretic framework and can be immediately applied to the conversion between automata and expressions. In Sec. 5 we derive a characterization of our new class by a finite set of forbidden minors. This implies that these minors represent the graph-structural properties of automata that cannot be encoded by regular expressions and thus cause the blowup observed in the construction of regular expressions from finite automata.

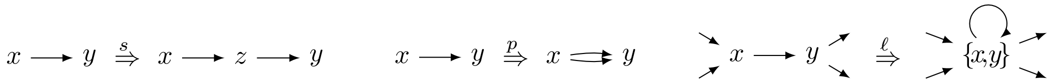
2 Preliminaries

We consider finite directed graphs with loops and multiple arcs. These are canonically known as *directed pseudographs* but will be referred to as just *graphs*. Formally, a graph is a tuple (V, A, t, h) with *vertices* V, *arcs* A, tail-map $t : A \to V$ and head-map $h : A \to V$. If G is not given explicitly, let $G = (V_G, A_G, t_G, h_G)$. An *xy-arc* of G is any $a \in A_G$ with $t_G(a) = x$ and $h_G(a) = y$; we write this as $a = xy \in A_G$. An xy-arc a *leaves* x and *enters* y, and x and y are called the *endpoints* of a. Distinct xy-arcs of a graph are *parallel* to each other. An xx-arc is an *x-loop* or just loop, every other arc is a *proper* arc. The *in-degree* of $x \in V_G$, denoted $\mathrm{d}^-_G(x)$, is the number of arcs entering x in G, the *out-degree* $\mathrm{d}^+_G(x)$ is the number of arcs leaving x. A *constriction* of G is any proper xy-arc where $\mathrm{d}^+_G(x) = 1 = \mathrm{d}^-_G(y)$. A vertex $x \in V_G$ is *simple* if $\mathrm{d}^-_G(x) \leq 1$ and $\mathrm{d}^+_G(x) \leq 1$. Subscripts are omitted if they are understood.

We write $F \subseteq G$ is F is a subgraph of G. If F and G are subgraphs of H and $a = xy \in A_H$ with $x \in V_F$ and $y \in V_G$, then a is called an (F, G)-arc, as well as an (x, G)- or an (F, y)-arc of H. A *path* of length n, denoted $\mathbf{P_n}$ is a graph on $n + 1$ vertices and n constrictions.

The *subdivision* of an arc $a = xy$ is the replacement of a with an xy-path of length 2. More generally, a subdivision of G, referred to as a DG, is any graph H s.t. there are graphs $G_1, \ldots, G_n$ where $G = G_1$, G_{i+1} results from subdividing an arc of G_i and $G_n = H$. The *split* of a vertex x is the replacement of x with two vertices x_1 and x_2 and an x_1x_2-arc and redirecting all arcs that entered x to enter x_1, resp. redirecting all arcs that left x to leave x_2. Two vertices $x, y \in V_G$ are *merged* by being replaced with a new vertex z and redirecting all arcs entering or leaving x or y to enter or leave z.

A graph G is *two-terminal* if there are $s, t \in V_G$ s.t. every $x \in V_G$ lies on some st-path in G. The vertices s and t are respectively called the *source* and *sink* of G; we write $G = (G, s, t)$ to express that G is two-terminal with source s and sink t. A two-terminal graph (G, s, t) is a *hammock* if $\mathrm{d}^-_G(s) = \mathrm{d}^+_G(t) = 0$. Let x and y be vertices of (G, s, t): x *dominates* y if x lies on every sy-path, and x *co-dominates* y if x lies on every yt-path. Furthermore, x is a *guard* of y if x dominates and co-dominates y; also, x is a guard of the arc a if x guards



(a) series expansion **(b)** parallel expansion **(c)** loop expansion

Figure 1 Expansions of an xy-arc, resp. the containing graph.

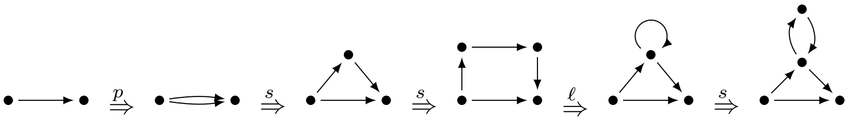


Figure 2 Construction of an spl-graph from $\mathbf{P_1}$ by a sequence of expansions.

both $t(a)$ and $h(a)$. More generally, x guards a subgraph F of (G, s, t) if x guards every arc and vertex of F.

3 SPL - Graphs

▶ **Definition 1.** The relations $\overset{s}{\Rightarrow}$, $\overset{p}{\Rightarrow}$ and $\overset{\ell}{\Rightarrow}$ are defined on graphs as follows: Let G be a graph and a an xy-arc in G, then

- $G \overset{s}{\Rightarrow} H$ if H is obtained by subdividing a in G
- $G \overset{p}{\Rightarrow} H$ if H is obtained from G by adding an arc which is parallel to a.
- $G \overset{\ell}{\Rightarrow} H$ if a is a constriction and H is obtained by merging x and y in G.

We say that H is derived from G by means of *series*-, *parallel*- or *loop*-expansion if $G \overset{s}{\Rightarrow} H$, $G \overset{p}{\Rightarrow} H$ or $G \overset{\ell}{\Rightarrow} H$, respectively. The local changes in G upon expansion are sketched in Fig. 1. We write $G \Rightarrow H$ if the particular expansion is irrelevant, and $G \Rightarrow^{\star} H$ if H is derived from G by a (possibly empty) finite sequence of expansions.

▶ **Definition 2.** The class of spl-graphs, denoted $\mathcal{SPL}$, is generated by $\Rightarrow$ from $\mathbf{P_1}$ as follows

- $\mathbf{P_1} \in \mathcal{SPL}$
- Let $G \in \mathcal{SPL}$, then $H \in \mathcal{SPL}$ if $G \overset{s}{\Rightarrow} H$ or $G \overset{p}{\Rightarrow} H$, or if $G \overset{\ell}{\Rightarrow} H$ where the ℓ-expanded arc is not incident to the source or the sink of G.

We call $\mathbf{P_1}$ the *axiom* of $\mathcal{SPL}$. The restriction imposed on ℓ-expansion ensures that every spl-graph is a hammock. An example for the step-wise construction of an spl-graph is shown in Fig. 2. The acyclic spl-graphs coincide with the arc-series-parallel graphs investigated by Valdes et al. [15]; we resort to their results whenever possible and elaborate only on properties of $\mathcal{SPL}$ that arise from its non-acyclic members.

To decide whether (G, s, t) is an spl-graph, we define a kind of dual to expansion. Some care must be taken with the removal of loops, which is why the new operations are restricted to hammocks.

▶ **Definition 3.** The relations $\overset{s}{\Leftarrow}$, $\overset{p}{\Leftarrow}$ and $\overset{\ell}{\Leftarrow}$ are defined on hammocks as follows: Let $G = (G, s, t)$ be a hammock, then

i) $G \overset{s}{\Leftarrow} H$ if y is simple vertex of G, incident to $a_1 = xy$ and $a_2 = yz$, and H is derived from G by removing y, a_1 and a_2 and adding an xz-arc.
ii) $G \overset{p}{\Leftarrow} H$ if H is derived from G by removing one of two parallel arcs.

iii) $G \overset{\ell}{\Leftarrow} H$ if a is an x-loop in G s.t. x does not guard any arc besides a, and H is the split of x in $G \setminus \{a\}$.

If $G \overset{\mathrm{c}}{\Leftarrow} H$ for $\mathrm{c} \in \{s, p, \ell\}$ we call both H and the replacement operation a c-*reduction* of G. As before we may simply write $G \Leftarrow H$ for any reduction, and $G \Leftarrow^{\star} H$ if H can be derived from G by a sequence of reductions.

Expansion and reduction are not proper duals as the latter relation is restricted to hammocks by definition. For hammocks we find $G \overset{\mathrm{c}}{\Leftarrow} H$ *iff* $H \overset{\mathrm{c}}{\Leftarrow} G$ for $\mathrm{c} \in \{s, p\}$; but while $G \overset{\ell}{\Leftarrow} H$ implies $H \overset{\ell}{\Rightarrow} G$, the converse is not true. The asymmetry is due to the fact that if ℓ-expansion introduces an x-loop a, x might guard some arc besides a, so the converse reduction is not ensured. This, however, does not happen within $\mathcal{SPL}$.

▶ **Proposition 4.** *Let C be a cycle of $G \in \mathcal{SPL}$. Then exactly one vertex of C guards C.*

The intuition of Prop. 4 is that every cycle in an spl-graph contains a vertex that serves as the unique 'entry' and 'exit' of this cycle wrt. the source and sink (see the last two steps in Fig. 2). Also note that a cycle might well be guarded by any number of vertices outside the cycle.

▶ **Theorem 5.** $G \in \mathcal{SPL}$ iff $G \Leftarrow^{\star} \mathbf{P_1}$

Proof. Since $G \Leftarrow^{\star} \mathbf{P_1}$ implies $\mathbf{P_1} \Rightarrow^{\star} G$, reducibility is sufficient for membership. Necessity is shown by induction on the structure of G. The claim holds for $\mathbf{P_1}$, so suppose $G \in \mathcal{SPL}$ where $G \Leftarrow^{\star} \mathbf{P_1}$ and let $G \Rightarrow H$. We attend to ℓ-expansion only, the other cases are trivial. Let $a = uv$ be the relevant constriction of G and let $l = xx$ be the loop introduced in H. If x guards some distinct arc $a' = yz$ in H, then G contains a cycle that defies Prop. 4, contrary to the assumption $G \in \mathcal{SPL}$. Therefore, $H \overset{\ell}{\Leftarrow} G$ is a valid reduction; since by assumption $G \Leftarrow^{\star} \mathbf{P_1}$, we find $H \Leftarrow^{\star} \mathbf{P_1}$. ◀

While membership in $\mathcal{SPL}$ can be decided by reducing a hammock to the axiom of $\mathcal{SPL}$, we do not know how to do so. Actually, there is no need for a strategy, since the reduction-system exhibits unique normal-forms. Using a standard argument from abstract rewriting (see e.g. [14]), we first show that reductions are locally confluent.

▶ **Lemma 6.** *Let G be a hammock and suppose $G \Leftarrow H_1$ and $G \Leftarrow H_2$ hold. Then there is a hammock J s.t. $H_1 \Leftarrow^{\star} J$ and $H_2 \Leftarrow^{\star} J$ hold.*

Each reduction decreases the number of arcs or loops and none introduces loops, so every sequence of reductions eventually terminates. Any graph derived from G by exhaustive reduction is called *normal-form* of G and denoted $\mathrm{R}(G)$. A graph G that coincides with its normal-form, $G = \mathrm{R}(G)$, is called *reduced.* Applying Newman's lemma [13, 14] yields

▶ **Corollary 7.** *The normal-form $\mathrm{R}(G)$ of any hammock G is unique.*

Computing the normal-form of a hammock can be realized by repeatedly running the reduction algorithm for arc-series-parallel graphs [15], interspersed with ℓ-reductions, until no further reduction can be applied. To this end some bookmarking about the loops occurring in the intermediate graphs is necessary. Testing whether x is a guard can be done in linear time by counting the components of $G \setminus x$. Overall, this method computes $\mathrm{R}(G)$ in quadratic time.

▶ **Theorem 8.** *Membership of G in $\mathcal{SPL}$ is effectively decidable due to*

$$G \in \mathcal{SPL} \quad \text{iff} \quad G \textit{ is a hammock and } \mathrm{R}(G) = \mathbf{P_1}$$

4 Encoding by Regular Expressions

Syntax and semantics of regular expressions (REs) follow Hopcroft & Ullman's textbook [8] except that we do not allow for $\emptyset$ in REs. As for notation, L_r denotes the language described by the RE r and $\mathrm{reg}(\Sigma)$ denotes the class of REs over Σ. An RE is *simplified* if it does not contain ε as a factor. Any RE r can be converted to a simplified RE $\mathrm{simp}(r)$, denoting the same language, by replacing every subexpression $s\varepsilon$ or εs with just s.

An *extended finite automaton* (EFA) over Σ is a 5-tuple $E = (Q, \Sigma, \delta, I, F)$, whose elements denote the set of states, the alphabet, the transition relation, the initial and the final states, respectively. These sets are all finite and satisfy $Q \cap \Sigma = \emptyset$, $\delta \subseteq Q \times \mathrm{reg}(\Sigma) \times Q$, $I \subseteq Q$, and $F \subseteq Q$. The relation $\vdash_E$ is defined on $Q \times \Sigma^*$ as $(q, ww') \vdash_E (q', w')$ if $(q, r, q') \in \delta$ and $w \in L_r$. The language accepted by E is

$$L(E) := \{w \mid (q_i, w) \vdash_E^* (q_f, \varepsilon) \text{ for } q_i \in I,\, q_f \in F\}$$

Two EFAs are *equivalent* if they accept the same language. An EFA is *normalized* if $|I| = |F| = 1$ and the initial and final state are distinct; any EFA can normalized by adding a new initial (final) state and ε-transitions from (to) this new initial (final) state to (from) the original ones. The EFA E is *trim* if for every state q of E there is a word $w = w_1w_2 \in L(E)$ s.t. $(q_i, w_1) \vdash_E^* (q, \varepsilon)$ and $(q, w_2) \vdash_E^* (q_f, \varepsilon)$ hold for some $q_i \in I$ and $q_F \in F$. Any EFA can be converted to a trim equivalent EFA by removing all states that do not meet this requirement and adjusting the transition relation. A *nondeterministic finite automaton* with ε-transitions (εNFA) is an EFA whose transition relation is restricted to $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$.

The graph *underlying* E is $G(E) := (Q, \delta, t, h)$ where $t : (p, r, q) \mapsto p$ and $h : (p, r, q) \mapsto q$. It is easy to see that E is trim and normalized *iff* $G(E)$ is a hammock.

An EFA displays a compromise between the complexity of its transition-labels and that of its underlying graph; REs and εNFAs represent the extremes in this tradeoff: an RE can be considered as an EFA whose underlying graph is trivial, namely $\mathbf{P_1}$, while an εNFA is an EFA with trivial labels. Locally relaying information about a language between the graph-structure of an EFA and its labels lies at the heart of several conversions between REs and εNFAs.

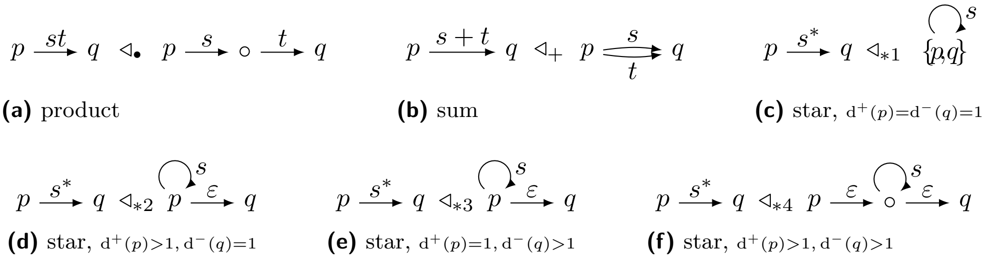
4.1 Expressions to Automata

We consider a fragment of the replacement-system proposed by Gulan & Fernau [7]. Let E be an EFA with transition $\tau = (p, r, q)$ where r contains operators, then τ can be replaced depending on the root of r, the out-degree of p and the in-degree of q. The degrees are only relevant if r is an iteration: in this case they determine whether p and q should be merged or a new state should be added (or neither) upon introduction of a loop. The rewriting rules, denoted $\triangleleft_\bullet$, $\triangleleft_+$, and $\triangleleft_{*1}$ to $\triangleleft_{*4}$ are shown in Fig. 3.

In order to convert an RE into an εNFA, we identify $r \in \mathrm{reg}(\Sigma)$ with the trivial EFA $A_r^0 := (\{q_i, q_f\}, \Sigma, \{(q_i, r, q_f)\}, \{q_i\}, \{q_f\})$, which obviously satisfies $L(A_r^0) = L_r$. The language accepted by an EFA is invariant under each rewriting, hence exhaustive application of $\triangleleft_\bullet$, $\triangleleft_+$ and $\triangleleft_{*i}$ yields a sequence $A_r^0, A_r^1, \ldots$ of equivalent EFAs terminating in an εNFA which we denote A_r.

▶ **Lemma 9.** *Every A_r^i satisfies $G(A_r^i) \in \mathcal{SPL}$.*

Thus the graph underlying A_r is an spl-graph, too. It is shown in [7] that A_r is unique. We thus define a map α from $\mathrm{reg}(\Sigma)$ to the class of expression-labeled spl-graphs (aka trim normalized EFAs) by setting $\alpha(r) := A_r$.


$p \xrightarrow{st} q \lhd_\bullet p \xrightarrow{s} \circ \xrightarrow{t} q$
$p \xrightarrow{s+t} q \lhd_+ p \; s, t \; q$
$p \xrightarrow{s^*} q \lhd_{*1} \{p,q\}$ s
$p \xrightarrow{s^*} q \lhd_{*2} p \xrightarrow{\varepsilon} q$ s
$p \xrightarrow{s^*} q \lhd_{*3} p \xrightarrow{\varepsilon} q$ s
$p \xrightarrow{s^*} q \lhd_{*4} p \xrightarrow{\varepsilon} \circ \xrightarrow{\varepsilon} q$ s

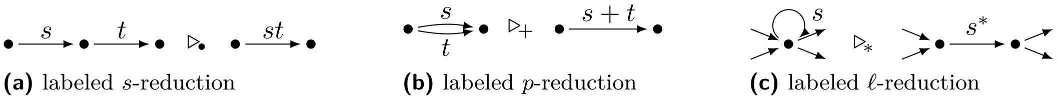
(a) product **(b)** sum **(c)** star, $d^+(p)=d^-(q)=1$

(d) star, $d^+(p)>1, d^-(q)=1$ **(e)** star, $d^+(p)=1, d^-(q)>1$ **(f)** star, $d^+(p)>1, d^-(q)>1$

Figure 3 Replacing a transition (p, r, q) based on its label and, in case $r = s^*$, the out-degree of p and the in-degree of q in $G(E)$. Either rule $\lhd_\bullet$, $\lhd_{*4}$ introduces a new state 'between' p and q.

4.2 Automata to Expressions

The spl-reductions are augmented to handle expression-labeled arcs, which yields a second rewriting-system on EFAs. In order to meet the requirements for loop-reduction, we consider normalized EFAs only. The labeled reductions, denoted $\rhd_\bullet$, $\rhd_+$, and $\rhd_*$, are shown in Fig. 4. Again, the accepted language is invariant under these transformations.



(a) labeled s-reduction **(b)** labeled p-reduction **(c)** labeled ℓ-reduction

Figure 4 Labeled spl-reductions

Exhaustive reduction of a normalized EFA E terminates in an equivalent EFA which we denote $\mathrm{R}_l(E)$. The graph underlying $\mathrm{R}_l(E)$ is the normal-form of the graph underlying E, $G(\mathrm{R}_l(E)) = \mathrm{R}(G(E))$, and we further find

▶ **Proposition 10.** *The labels of* $\mathrm{R}_l(E)$ *are unique up to associativity and commutativity.*

In particular if $G(E) \in \mathcal{SPL}$, we find $G(\mathrm{R}_l(E)) = \mathbf{P_1}$, so the only label of $\mathrm{R}_l(E)$ is an RE r with $L_r = L(E)$. By Prop. 10 this RE is unique up to trivialities, so we define a map β from EFAs with spl-structure to REs by setting $\beta(E) := r$, where r is the label of $\mathrm{R}_l(E)$.

4.3 Duality of the Conversions

The conversions between REs and εNFAs with spl-structure are 'almost' duals, some extra effort arises with the treatment of ε-factors resp. certain ε-labeled transitions. This is due to the fact that star-expansion might introduce ε-transition that have no corresponding subterm in the

We write $r = r'$ if the expressions r and r' are identical up to associativity and commutativity of the regular operators.

▶ **Theorem 11.**

1. $\mathrm{simp}(r) = \beta(\alpha(\mathrm{simp}(r)))$ *for any RE* r
2. $A = \alpha(\mathrm{simp}(\beta(A)))$ *for any* ε*NFA* A *with* $G(A) \in \mathcal{SPL}$

Thus the encoding of labeled spl-graphs by simplified expressions is unique and reversible, and every simplified expression encodes a labeled spl-graph. Hence every RE over a non-empty alphabet encodes an spl-graph and every spl-graph can be encoded. Informally, we state

▶ **Corollary 12.** $G \in \mathcal{SPL}$ iff G *can be encoded by an RE*

5 Forbidden Minor Characterization

We adapt the notion of topological minors, which is well-known for undirected graphs (see e.g. [2]), to our needs.

▶ **Definition 13.** An *embedding* of F in G is an injection $e : V_F \to V_G$ satisfying that if $a = xy \in A_F$, then G contains an $e(x)e(y)$-path P_a, and that P_a and $P_{a'}$ are internally disjoint for distinct $a, a' \in A_F$.

If an embedding of F in G exists, we call F a *minor* of G *realized* by the embedding. We write $F \preccurlyeq G$ if F is a minor of G. If $F \preccurlyeq G$ does not hold then G is F*-free*; if $\mathcal{M}$ is a set of graphs and G is F-free for every $F \in \mathcal{M}$, then G is $\mathcal{M}$-free. It is easily seen that subdivisions allow for an equivalent characterization of minors:

▶ **Proposition 14.** $F \preccurlyeq G$ iff G *contains a* $\mathrm{D}F$

Let $F \preccurlyeq G$ be realized by e and $x \in V_F$, we call $e(x)$ a *peg* of F in G wrt. e; if G and e are known, we omit mentioning them. Observe that the in-/out-degree of a vertex in F does not exceed the in-/out-degree of its corresponding peg in G:

▶ **Proposition 15.** *If e realizes $F \preccurlyeq G$, then* $\mathrm{d}^-_F(x) \leq \mathrm{d}^-_G(e(x))$ *and* $\mathrm{d}^+_F(x) \leq \mathrm{d}^+_G(e(x))$.

Let e realize $F \preccurlyeq G$, a *bypass* of F in G wrt. e is an $e(x)e(y)$-path in G, where xy is *not* an arc of F. An embedding of F in G is *bare* if G contains no bypass of F wrt. to the embedding; we then write $M \sqsubseteq G$. Observe that $F \preccurlyeq G$ might well be realized by various — in particular bare and non-bare — embeddings. Based on Prop. 14, we also call a $\mathrm{D}F$ in G bare if G contains no bypass wrt. to the embedding realizing this $\mathrm{D}F$.

The existence of an xy-path is invariant under spl-expansion and -reduction if x and y are not subject to the operation.

▶ **Proposition 16.** *Let $G \Rightarrow H$ or $G \Leftarrow H$ and $\{x, y\} \subseteq V_G \cap V_H$, then G contains an xy-path* iff *H does.*

'Half' of the sought characterization is given by the set of graphs $\boldsymbol{\mathcal{F}} = \{\mathbf{C}, \mathbf{C}^{\mathrm{R}}, \mathbf{N}, \mathbf{Q}\}$, shown in Fig. 5. Note that Valdes et al. proved that an acyclic hammock is arc-series-parallel *iff* it is $\mathbf{N}$-free [15].

▶ **Lemma 17.** *Every $G \in \mathcal{SPL}$ is $\boldsymbol{\mathcal{F}}$-free.*

Proof. Clearly, $\mathbf{P_1}$ is $\boldsymbol{\mathcal{F}}$-free. Assume $G \in \mathcal{SPL}$ is $\boldsymbol{\mathcal{F}}$-free and let $G \Rightarrow H$. Consider any $F \in \boldsymbol{\mathcal{F}}$: since F is free of parallel arcs, and the existence of paths among vertices in $V_G \cap V_H$ is invariant under expansion (Prop. 16), $F \preccurlyeq H$ implies that a peg of F in H was introduced upon expansion. Hence in case $G \overset{p}{\Rightarrow} H$, F is not a minor of H, i.e., H is $\boldsymbol{\mathcal{F}}$-free. The same goes for $G \overset{s}{\Rightarrow} H$: as the new vertex in H is simple, but no vertex of F is, Prop. 15 implies that H is F-free and therefore $\boldsymbol{\mathcal{F}}$-free.
If $G \overset{\ell}{\Rightarrow} H$, let $a = xy$ be the relevant constriction of G and $l = zz$ the loop of H introduced by expansion. If $F \preccurlyeq H$ is realized by e, then $z = e(q)$ for some $q \in V_F$, as was discussed above. Let $H' = H \setminus l$: since F is free of loops, $F \preccurlyeq H'$ holds, too, and since q is not simple in F, z is not simple in H'. We actually find $\mathrm{d}^-_{H'}(z) \geq 2$ *and* $\mathrm{d}^+_{H'}(z) \geq 2$: if $\mathrm{d}^-_{H'}(z) = 0$, then $F \preccurlyeq G$ is realized by e', which is defined as e except that $e'(q) = y$ — contradicting the assumption that G is $\boldsymbol{\mathcal{F}}$-free. If $\mathrm{d}^-_{H'}(z) = 1$, there is exactly one arc entering z in H. Let

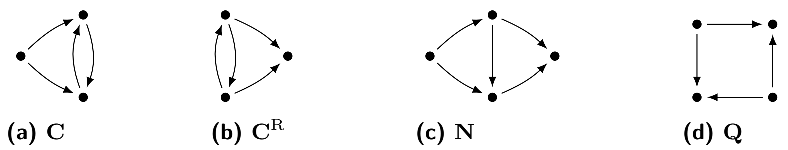

(a) $\mathbf{C}$ (b) $\mathbf{C}^{\mathrm{R}}$ (c) $\mathbf{N}$ (d) $\mathbf{Q}$

Figure 5 Graphs constituting $\boldsymbol{\mathcal{F}}$.

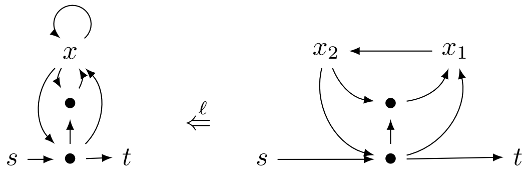


Figure 6 The graph $\mathbf{N}$ emerges as a subgraph due to ℓ-reduction of a hammock. Still, $\mathbf{C}$ is a minor of either side.

this be $a' = z'z$, then $F \preccurlyeq G$ is realized by e'' which is as e except that, again, $e''(q) = y$, contradicting our assumption. A symmetric argument shows $\mathrm{d}^{+}_{H'}(z) \geq 2$. In fact, we have also shown that q, of which z is the peg, has in- and out-degree at least two.
But since $\mathrm{d}^{-}_{G}(x) = \mathrm{d}^{-}_{H'}(z)$ and $\mathrm{d}^{+}_{G}(y) = \mathrm{d}^{+}_{H'}(z)$, some $F' \in \boldsymbol{\mathcal{F}}$, constructed by splitting q in F satisfies $F' \preccurlyeq G$ — contradicting the assumption that G is $\boldsymbol{\mathcal{F}}$-free. ◀

Likewise, it can be shown in general that if $H \Leftarrow G$ and H is not $\boldsymbol{\mathcal{F}}$-free, then neither is G. However, there is a catch: the $\boldsymbol{\mathcal{F}}$-minors of G and H need not coincide. This is hinted at by the following lemma, and an explicit example is shown in Fig. 6.

▶ **Lemma 18.** *If $H \Leftarrow G$ for hammocks H and G, then H is $\boldsymbol{\mathcal{F}}$-free* iff *G is. More specifically:*

i) $F \preccurlyeq H$ iff $F \preccurlyeq G$ *for* $F \in \{\mathbf{C}, \mathbf{C}^{\mathrm{R}}, \mathbf{Q}\}$
ii) $\mathbf{N} \preccurlyeq H$ *only if* $\mathbf{N} \preccurlyeq G$*, whereas*
iii) $\mathbf{N} \preccurlyeq G$ *only if* $(\mathbf{N} \preccurlyeq G$ *or* $\mathbf{C} \preccurlyeq G$ *or* $\mathbf{C}^{\mathrm{R}} \preccurlyeq G)$

Still, $\boldsymbol{\mathcal{F}}$-freeness of a hammock is not sufficient for membership in $\mathcal{SPL}$: for example, the hammock $\boldsymbol{\Phi}$, shown in Fig. 7a, is $\boldsymbol{\mathcal{F}}$-free, but not included in $\mathcal{SPL}$. The additional graphs necessary for a characterization by forbidden minors are $\boldsymbol{\Phi}$, $\boldsymbol{\Psi}$, and $\boldsymbol{\Psi}^{\mathrm{R}}$, shown in Fig. 7.

▶ **Lemma 19.** *Every $G \in \mathcal{SPL}$ is free of bare $\boldsymbol{\Phi}$-, $\boldsymbol{\Psi}$-, and $\boldsymbol{\Psi}^{\mathrm{R}}$-minors*

On the other hand, each of $\{\boldsymbol{\Phi}, \boldsymbol{\Psi}, \boldsymbol{\Psi}^{\mathrm{R}}\}$ may well be a minor of certain spl-graphs. Examples for spl-graphs with $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ as minors are given in Fig. 8, the reader might want to check that these can be reduced to $\mathbf{P_1}$. In the absence of $\boldsymbol{\mathcal{F}}$-minors an invariance-result akin to Lem. 18 holds for bare subdivisions of these three graphs.

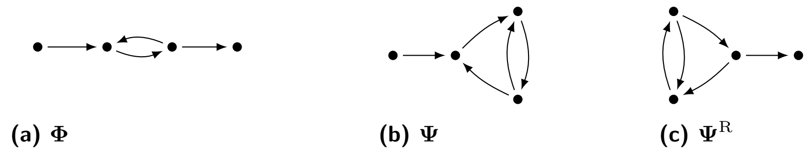

(a) $\boldsymbol{\Phi}$ (b) $\boldsymbol{\Psi}$ (c) $\boldsymbol{\Psi}^{\mathrm{R}}$

Figure 7 Graphs that do not allow for a bare embedding in any $G \in \mathcal{SPL}$.

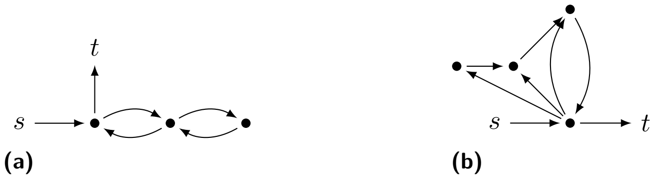


Figure 8 Examples for spl-graphs with $\mathbf{\Phi}$ and $\mathbf{\Psi}$ as (non-bare) minors

▶ **Lemma 20.** *Let H be an $\boldsymbol{\mathcal{F}}$-free hammock and assume $H \Leftarrow G$, then $F \sqsubseteq H$* iff *$F \sqsubseteq G$ for $F \in \{\mathbf{\Phi}, \mathbf{\Psi}, \mathbf{\Psi}^{\mathrm{R}}\}$.*

Proof. Each of $\mathbf{\Phi}$, $\mathbf{\Psi}$, and $\mathbf{\Psi}^{\mathrm{R}}$ is free of parallel arcs, so Prop. 16 yields the claim if all pegs occur in $V_G \cap V_H$; in particular, nothing needs to be done for $H \overset{p}{\Leftarrow} G$. We prove the claim for $\mathbf{\Phi}$, the procedure is the same for $\mathbf{\Psi}$ and $\mathbf{\Psi}^{\mathrm{R}}$. In the following, let H be $\boldsymbol{\mathcal{F}}$-free.
Let $\mathbf{\Phi} \sqsubseteq H$ be realized by e. If $G \overset{s}{\Leftarrow} H$ removes a peg $x = e(q)$, q is one of the two simple vertices of $\mathbf{\Phi}$; here, let q be the unique vertex with $\mathrm{d}^-_{\mathbf{\Phi}}(q) = 0$ (the other case is symmetric). Since s-reduction is applicable due to x, an arc $a = yx$ exists in H, with y also occurring in G. Let e' be an embedding of $\mathbf{\Phi}$ in G, s.t. $e'(q) = y$ and e' as e for the other vertices. If e' is bare, the claim follows for $\mathbf{\Phi}$ and s-reduction, so assume it is not. Then G contains a bypass of $\mathbf{\Phi}$ wrt. e', which is necessarily a path *leaving* y, otherwise H would contain a bypass of $\mathbf{\Phi}$ wrt. e, contradicting the assumption that e is bare. We find $\mathbf{C} \preccurlyeq G$, if the other endpoint of the bypass is the peg of the vertex in $\mathbf{\Phi}$'s cycle that is not adjacent to q. If the bypass is from $e'(q)$ to the peg of the vertex with out-degree 0 in $\mathbf{\Phi}$, we get $\mathbf{Q} \preccurlyeq G$. In both cases Lem. 18 implies that H is not $\boldsymbol{\mathcal{F}}$-free, contradicting our assumption. Proving that s-reduction does not introduce new bare D$\mathbf{\Phi}$'s is trivial.
Again let $\mathbf{\Phi} \sqsubseteq H$ be realized by e with peg $x \in V_H$. Considering $H \overset{\ell}{\Leftarrow} G$, let $a = xx$ be the loop that allows for reduction, and let x_1x_2 denote that constriction arising from it. As in the proof of Lem. 18 our argument is based on the facts that a is irrelevant for the D$\mathbf{\Phi}$ in H and that $\mathrm{d}^-_G(x_1) = \mathrm{d}^-_{H\setminus a}(x)$ and $\mathrm{d}^+_G(x_2) = \mathrm{d}^-_{H\setminus a}(x)$ hold. Since every of $\mathbf{\Phi}$ has either in- or out-degree ≤ 1, we can construct an embedding e' of $\mathbf{\Phi}$ in G by assigning the role of x to either x_1 or x_2. ◀

▶ **Definition 21.** A *kebab* is a connected graph consisting of three arc-disjoint subgraphs: a strong component B, called the *body*, and two nonempty vertex-disjoint paths S_1 and S_2, called the *spikes* of the kebab.

We name some unique vertices in a kebab: the endpoint of a spike connecting that spike to the body is the *puncture* of this spike, the other endpoint is its *tip*. A spike which enters the body of a kebab is an *in-spike*, one that leaves the body is called an *out-spike*. If both spikes of a kebab K enter (leave) the body, K is also called an *in*-kebab (*out*-kebab) if one enters and the other leaves the body, K is called an *inout*-kebab. In order to prove two lemmas concerning kebabs, some auxilliary propositions are necessary.

▶ **Proposition 22.** *Let G be a reduced hammock with distinct arcs a_1, a_2 s.t. $h(a_1) = v = t(a_2)$. Then v is incident to a third proper arc.*

▶ **Proposition 23.** *Let x and y be distinct vertices of a hammock G. Then exactly one of the following is true: 1) x dominates y, 2) y dominates x, or 3) for some $z \in V_G \setminus \{x, y\}$, G contains internally disjoint zx- and zy-paths.*

▶ **Proposition 24.** *Let x and y be distinct vertices of a strong graph G, then there is a cycle $C \subseteq G$ and distinct $z_x, z_y \in V_C$, s.t. G contains an xz_x- and a yz_y-path that are disjoint.*

▶ **Lemma 25.** *Let (G, s, t) be an spl-reduced hammock and suppose G contains a kebab. Then $F \preccurlyeq G$ for some $F \in \boldsymbol{\mathcal{F}}$ or $\mathbf{\Phi} \sqsubseteq G$.*

Proof. We choose a 'biggest' kebab $K \subseteq G$ with the following properties

1. the body of K is arc-maximal, i.e., no kebab of G has a body with more arcs
2. the spikes of K are inclusion-maximal in G, i.e., they are not 'sub-spikes' of a bigger kebab with the same body as K but longer spikes than K.

We need to distinguish whether K is in an in-, an out- or an inout-kebab. Due to space restrictions we only treat the first case, however note that the first and second case are symmetric.

Let $K \subseteq G$ be an in-kebab and let B denote the body of K, S_1 and S_2 the spikes, with tips t_1 and t_2, and punctures p_1 and p_2, respectively (Fig. 9a). As (G, s, t) is a hammock, according to Prop. 23 either one of t_1 and t_2 dominates the other, or G contains a vertex x and internally disjoint xt_1- and xt_2-paths.

1. If t_2 dominates t_1 (the converse case is symmetric), let P be a shortest t_2t_1-path in G. If P and B are disjoint, then P contains a segment P' from S_2 to S_1. Using Prop. 24 we now find $\mathbf{C} \preccurlyeq G$ (Fig. 9b). So let P go through B, then the last segment of P is a (B, t_2)-path outside B. By the choice of K and P this segment consist of a single arc $a = bt_1$ for $b \in V_B$ (Fig. 9c). According to Prop. 22, t_1 is incident to a further arc a', as G is reduced. Our choice of K requires that the other endpoint z of a' lies in K, since B, S_1 and a form a strong component bigger than B. It is now easy to see (from Fig. 9c), that $z \in V_{S_2}$ yields $\mathbf{C} \preccurlyeq G$ (regardless of a's orientation), and that $z \in V_B$ yields $\mathbf{C} \preccurlyeq G$ or $\mathbf{C}^{\mathrm{R}} \preccurlyeq G$ (depending on a's orientation), so let $z \in V_{S_1} \setminus \{p_1\}$. This leaves two possibilities: If $a' = zt_1$ we find $\mathbf{Q} \preccurlyeq G$, with pegs t_1,p_1,b and z (Fig. 9d). On the other hand, $a' = t_1z$ leads to a contradiction: Since G is p-reduced, there is at least one vertex z' between t_1 and z on S_1; omitting the t_1z'-segment of S_1 lets us identify an in-kebab with tips z' and t_2 and a body properly containing B (Fig. 9e), contradicting maximality of B.

2. Let G contain a zt_1-path P_1 and a zt_2-path P_2 which are internally disjoint. If both P_i are disjoint with B, we find $\mathbf{C} \preccurlyeq G$ with help of Prop. 24 (Fig. 9f, where x_i denotes the first vertex on P_i that is also in S_i). If wlog. P_1 intersects B, let b denote the last vertex on P_1 that is in B and x the first vertex on P_1 that is in $V_{S_i} \setminus \{p_i\}$. If $x \neq t_1$, we find a kebab in G with a body containing B, contradicting our choice of K. As the claim was already proven for $x = t_1$ (see Fig. 9c), the statement follows for in-kebabs.

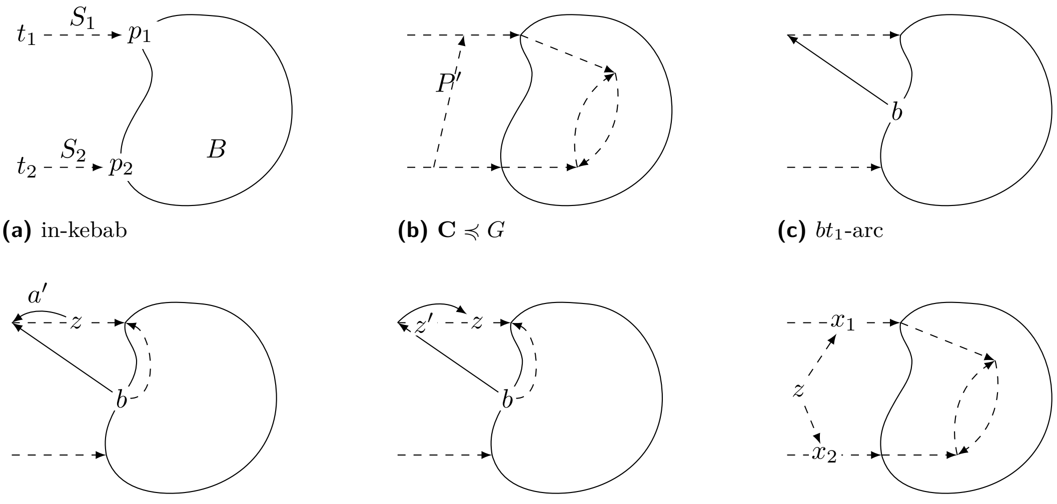
◀

▶ **Lemma 26.** *Let $G \neq \mathbf{P_1}$ be a reduced hammock with cycles. Then $F \preccurlyeq G$ for some $F \in \boldsymbol{\mathcal{F}}$ or $F' \sqsubseteq G$ for some $F' \in \{\mathbf{\Phi}, \mathbf{\Psi}, \mathbf{\Psi}^{\mathrm{R}}\}$.*

We have thus found a characterization of $\mathcal{SPL}$ by forbidden subgraphs.

▶ **Theorem 27.** *Let G be a hammock, then*

$$G \in \mathcal{SPL} \quad \text{iff} \quad G \textit{ is } \boldsymbol{\mathcal{F}}\textit{-free and no } F' \in \{\mathbf{\Phi}, \mathbf{\Psi}, \mathbf{\Psi}^{\mathrm{R}}\} \textit{ is a bare minor of } G.$$



(a) in-kebab **(b)** $\mathbf{C} \preccurlyeq G$ **(c)** bt_1-arc

(d) $\mathbf{Q} \preccurlyeq G$ **(e)** $a' = t_1 z$ **(f)** $\mathbf{C} \preccurlyeq G$

Figure 9 Cases occurring in the proof of Lem. 25 for K being an in-kebab. Solid arrows represent arcs, dashed arrows represent paths.

Proof. Let $G \in \mathcal{SPL}$, then Lem. 17 states that G is $\boldsymbol{\mathcal{F}}$-free, while Lem. 19 states that none of $\{\boldsymbol{\Phi}, \boldsymbol{\Psi}, \boldsymbol{\Psi}^{\mathrm{R}}\}$ is a bare minor of G. Conversely if $G \notin \mathcal{SPL}$ then Cor. 7 yields $\mathrm{R}(G) \neq \mathbf{P_1}$. By Valdes' result and Lem. 26, we know $F \preccurlyeq \mathrm{R}(G)$ for some $F \in \boldsymbol{\mathcal{F}}$ and/or $F' \sqsubseteq \mathrm{R}(G)$ for some $F' \in \{\boldsymbol{\Phi}, \boldsymbol{\Psi}, \boldsymbol{\Psi}^{\mathrm{R}}\}$. If $G = \mathrm{R}(G)$, i.e., G is already reduced, the claim follows immediately; otherwise, induction on the length of the reduction using Lems. 18 and 20 provides the statement. ◀

6 Conclusions

We generalized the class of arc-series-parallel graphs by augmenting the standard construction with a rule that allows for loops. Members of the new class can be reversibly encoded by regular expressions which represent the recursive structure of a graph; naturally, the size of such an encoding is linear in that of the input. Moreover, any regular expression represents an spl-graph under this encoding. Modulo isomorphism of graphs, resp. modulo associativity and commutativity of operators in expressions, the encoding and decoding are unique; thus they provide — up to trivialities — a bijection between spl-graphs and regular expressions.

The encoding is done by constructing a series of arc-labeled spl-graphs. As an automaton can be interpreted as an arc-labeled graph, this can be immediately applied to the conversion of finite automata with spl-structure to equivalent regular expressions whose size is linear wrt. to the automaton. This generalizes a recent result for acyclic automata.

We further characterized our new class by means of 7 forbidden minors. Therefore the exponential increase of expression size over automaton size, which cannot be avoided in the general case, is due to graph-structural properties of automata that are not present in spl-graphs. The forbidden minors can be considered as being the primitives of these non-expressible properties, they should be further investigated in order to improve on current conversions from automata to expressions.

References

1 Marek Chrobak. Finite automat and unary languages. *Theoretical Computer Science*, (47):149–158, 1986.

2 Reinhard Diestel. *Graph Theory.* Number 173 in Graduate Texts in Mathematics. Springer, 4 edition, 2010.

3 A. Ehrenfeucht and P. Zeiger. Complexity measures for regular expressions. *Journal of Computer and System Sciences*, 12:134–146, 1976.

4 K. Ellul, B. Krawetz, J. Shallit, and M.-W. Wang. Regular expressions: new results and open problems. *Journal of Automata, Languages and Combinatorics*, 10(4):407–437, 2005.

5 H. Gruber and M. Holzer. Finite automat, digraph connectivity, and regular expression size. In *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 39–50, 2008.

6 S. Gulan and H. Fernau. Local elimination-strategies in finite automata for shorter regular expressions. In *SOFSEM 2008*, volume 2, pages 46–57, 2008.

7 S. Gulan and H. Fernau. An optimal construction of finite automata from regular expressions. In R. Hariharan et al., editor, *28th FSTTCS*, pages 211–222, 2008.

8 John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

9 S. C. Kleene. *Representation of Events in Nerve Nets and Finite Automata*, pages 3–41. Annals of Mathematics Studies. 1956.

10 M. Korenblit and V. E. Levit. On algebraic expressions of series-parallel and fibonacci graphs. In C. S. Calude et al., editor, *DMTCS 2003*, number 2731 in LNCS, pages 215–224, 2003.

11 R. McNaughton. Techniques for manipulating regular expressions. In J. F. Hart and S. Takasu, editors, *Systems and Computer Science*, pages 27–41. University of Toronto Press in association with the University of Western Ontario Toronto, 1967.

12 Nelma Moreira and Rogério Reis. Series-parallel automata and short regular expressions. *Fundamenta Informaticae*, 91(3-4):611–629, 2009.

13 M.H.A. Newman. On theories with a combinatorial definition of ”equivalence”. *Annals of Mathematics*, 43(2), 1942.

14 Enno Ohlebusch. *Advanced Topics in Term Rewriting.* Springer, 2002.

15 Jacobo Valdes, Robert E. Tarjan, and Eugene L. Lawler. The recognition of series parallel digraphs. *SIAM Journal on Computing*, 11(2):298–313, 1981.

16 B. W. Watson. A taxonomy of finite automata construction algorithms. Technical Report Computing Science Note 93/43, Eindhoven University of Technology, 1994.

Extended Regular Expressions: Succinctness and Decidability

Dominik D. Freydenberger

**Institut für Informatik, Goethe Universität,
Frankfurt am Main, Germany**
`freydenberger@em.uni-frankfurt.de`

Abstract

Most modern implementations of regular expression engines allow the use of variables (also called back references). The resulting extended regular expressions (which, in the literature, are also called practical regular expressions, rewbr, or regex) are able to express non-regular languages.

The present paper demonstrates that extended regular-expressions cannot be minimized effectively (neither with respect to length, nor number of variables), and that the tradeoff in size between extended and "classical" regular expressions is not bounded by any recursive function. In addition to this, we prove the undecidability of several decision problems (universality, equivalence, inclusion, regularity, and cofiniteness) for extended regular expressions. Furthermore, we show that all these results hold even if the extended regular expressions contain only a single variable.

1998 ACM Subject Classification F.4.3: Formal Languages

Keywords and phrases extended regular expressions, regex, decidability, non-recursive tradeoffs

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.507

1 Introduction

Since being introduced by Kleene [18] in 1956, regular expressions have developed into a central device of theoretical and applied computer science. On one side, research into the theoretical properties of regular expressions, in particular various aspects of their complexity, is still a very active area of investigation (see Holzer and Kutrib [16] for a survey with numerous recent references). On the other side, almost all modern programming languages offer regular expression matching in their standard libraries or application frameworks, and most text editors allow the use of regular expressions for search and replacement functionality.

But, due to practical considerations (cf. Friedl [13]), most modern matching engines have evolved to use an extension to regular expressions that allows the user to specify non-regular languages. In addition to the features of regular expressions as they are mostly studied in theory (which we, from now on, call *proper regular expressions*), and apart from the (regularity preserving) "syntactic sugar" that most implementations use, these *extended regular expressions* contain *back references*, also called *variables*, which specify repetitions that increase the expressive power beyond the class of regular languages. For example, the (non-regular) language $L = \{ww \mid w \in \{\mathtt{a},\mathtt{b}\}^*\}$ is generated by the extended regular expression $\alpha := \big((\mathtt{a} \mid \mathtt{b})^*\big)\%x\, x$.

This expression can be understood as follows (for a more formal treatment, see Definition 3): For any expression β, $(\beta)\%x$ matches the same expression as β, and binds the match to the variable x. In the case of this example, the subexpression $\big((\mathtt{a} \mid \mathtt{b})^*\big)\%x$ can be matched to any word $w \in \{\mathtt{a},\mathtt{b}\}^*$, and when it is matched to w, the variable x is assigned the value w. Any further occurrence of x repeats w, leading to the language of all words of

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 507–518
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the form ww with $w \in \{\mathtt{a}, \mathtt{b}\}^*$. Analogously, the expression $\big((\mathtt{a} \mid \mathtt{b})^*\big)\,\%x\; xx$ generates the language of all www with $w \in \{\mathtt{a}, \mathtt{b}\}^*$.

Although this ability to specify repetitions is used in almost every modern matching engine (e. g., the programming languages PERL and Python), the implementations differ in various details, even between two versions of the same implementation of a programming language (for some examples, see Câmpeanu and Santean [6]). Nonetheless, there is a common core to these variants, which was first formalized by Aho [1], and later by Câmpeanu, Salomaa and Yu [5]. Still, theoretical investigation of extended regular expressions has been comparatively rare (in particular when compared to their more prominent subclass); see e. g., Larsen [20], Della Penna et al. [12], Câmpeanu and Santean [6], Carle and Narendran [8], and Reidenbach and Schmid [22].

In contrast to their widespread use in various applications, extended regular expressions have some undesirable properties. Most importantly, their *membership problem* (the question whether an expression matches a word) is NP-complete (cf. Aho [1]); the exponential part in the best known upper bounds depends on the number of different variables in the expression. Of course, this compares unfavorably to the efficiently decidable membership problem of proper regular expressions (cf. Aho [1]). On the other hand, there are cases where extended regular expressions express regular languages far more succinctly than proper regular expressions. Consider the following example:

▶ **Example 1.** For $n \geq 1$, let $L_n := \{www \mid w \in \{\mathtt{a}, \mathtt{b}\}^+, |w| = n\}$. These languages L_n are finite, and hence, regular. With some effort[1], one can prove that every proper regular expression for L_n is at least of length exponential in n. In contrast to this, L_n is generated by the extended regular expression

$$\alpha_n := (\underbrace{(\mathtt{a} \mid \mathtt{b}) \cdots (\mathtt{a} \mid \mathtt{b})}_{n \text{ times } (\mathtt{a} \mid \mathtt{b})})\%x\; xx,$$

which is of a length that is linear in n. ◇

Due to the repetitive nature of the words of languages L_n in Example 1, it is not surprising that the use of variables provides a shorter description of L_n. The following example might be considered less straightforward:

▶ **Example 2.** Consider the expression $\alpha := (\mathtt{a} \mid \mathtt{b})^+((\mathtt{a} \mid \mathtt{b})^+)\%x\, x(\mathtt{a} \mid \mathtt{b})^+$. It is a well-known fact that every word $w \in \{\mathtt{a}, \mathtt{b}\}^*$ with $|w| \geq 4$ can be expressed in the form $w = uxxv$, with $u, v \in \{\mathtt{a}, \mathtt{b}\}^*$ and $x \in \{\mathtt{a}, \mathtt{b}\}^+$ (as is easily verified by examining all four letter words). Thus, the expression α matches all but finitely many words; hence, its language $L(\alpha)$ is regular. ◇

The phenomenon used in Example 2 is strongly related to the notion of *avoidable patterns* (cf. Cassaigne [9]), and involves some very hard combinatorial questions. We observe that extended regular expressions can be used to express regular languages more succinctly than proper regular expressions do, and that it might be hard to convert an extended regular expression into a proper regular expression for the same language.

The two central questions studied in the present paper are as follows: First, how hard is it to minimize extended regular expressions (both with respect to their length, and with respect to the number of variables they contain), and second, how succinctly can extended

[1] One can show this by proving that every NFA for L_n requires at least $O(2^n)$ states, e. g., by using the technique by Glaister and Shallit [14]. Due to the construction used in the proof of Theorem 2.3 in [17], this also gives a lower bound on the length of the regular expressions for L_n.

regular expressions describe regular languages? These natural questions are also motivated by practical concerns: If a given application reuses an expression many times, it might pay off to invest resources in the search for an expression that is shorter, or uses fewer variables, and thus can be matched more efficiently.

We approach this question through related decidability problems (e. g., the universality problem) and by studying lower bounds on the tradeoff between the size of extended regular expressions and proper regular expressions.

The main contribution of the present paper is the proof that all these decision problems are undecidable (some are not even semi-decidable), even for extended regular expressions that use only a single variable. Thus, while bounding the number of variables in extended regular expressions (or, more precisely, the number of variable bindings) reduces the complexity of the membership problem from NP-complete to polynomial (cf. Aho [1]), we show that extending proper regular expressions with only a single variable already results in undecidability of various problems.

As a consequence, extended regular expressions cannot be minimized effectively, and the tradeoff between extended and proper regular expressions is not bounded by any recursive function (a so-called *non-recursive tradeoff*, cf. Kutrib [19]). Thus, although the use of the "right" extended regular expression for a regular expression might offer arbitrary advantages in size (and, hence, parsing speed), these optimal expressions cannot be found effectively. These results highlight the power of the variable mechanism, and demonstrate that different restrictions than the number of variables ought to be considered.

The structure of the further parts of the paper is as follows: In Section 2, we introduce most of the technical details. Section 3 consists of Theorem 9 – the main undecidability result – and its consequences, while Section 4 contains the proof of Theorem 9 and technical groundwork needed for that proof. The paper is concluded by Section 5. Due to space reasons, most technical details were omitted from the present version.

2 Preliminaries

This paper is largely self-contained. Unexplained notions can be found in Hopcroft and Ullman [17], Cutland [11], and Minsky [21].

2.1 Basic Definitions

Let $\mathbb{N}$ be the set of natural numbers, including 0. The function div denotes integer division, and mod denotes its remainder (e. g., $5 \operatorname{div} 3 = 1$ and $5 \bmod 3 = 2$). We denote the *empty string* by λ. For the *concatenation* of two strings w_1 and w_2, we write $w_1 \cdot w_2$ or simply $w_1 w_2$. We say a string $v \in A^*$ is a *factor* of a string $w \in A^*$ if there are $u_1, u_2 \in A^*$ such that $w = u_1 v u_2$. The notation $|K|$ stands for the size of a set K or the length of a string K.

If A is an alphabet, a *(one-sided) infinite word over* A is an infinite sequence $w = (w_i)_{i=0}^{\infty}$ with $w_i \in A$ for every $i \geq 0$. We denote the set of all one-sided infinite words over A by A^ω and, for every $a \in A$, let a^ω denote the word $w = (w_i)_{i=0}^{\infty}$ with $w_i = a$ for every $i \geq 0$. We shall only deal with infinite words $w \in A^\omega$ that have the form $w = u\, a^\omega$ with $u \in A^*$ and $a \in A$. Concatenation of words and infinite words is defined canonically: For every $u \in A^*$ and every $v \in A^\omega$ with $v = (v_i)_{i=0}^{\infty}$, $u \cdot v := w \in A^\omega$, where $w_0 \cdot \ldots \cdot w_{|u|-1} = u$ and $w_{i+|u|} = v_i$ for every $i \geq 0$, while vu is undefined. In particular, note that $a\, a^\omega = a^\omega$ for every $a \in A$.

2.2 Extended Regular Expressions

We now introduce syntax and semantics of extended regular expressions. Apart from some changes in terminology, this formalization is due to Aho [1]:

▶ **Definition 3.** Let Σ be an infinite set of *terminals*, let X be an infinite set of *variables*, and let the set of *metacharacters* consist of λ, (,), |, *, and %, where all three sets are pairwise disjoint. We define *extended regular expressions* inductively as follows:

1. Each $a \in \Sigma \cup \{\lambda\}$ is an extended regular expression that matches the word a.
2. Each $x \in X$ is an extended regular expression that matches the word to which x is bound.
3. If α_1 and α_2 are extended regular expressions, then $(\alpha_1 \mid \alpha_2)$ is an extended regular expression that matches any word matched by α_1 or by α_2.
4. If α_1 and α_2 are extended regular expressions, then $(\alpha_1\alpha_2)$ is an extended regular expression that matches any word of the form vw, where v matches α_1 and w matches α_2.
5. If α is an extended regular expression, then $(\alpha)^*$ is an extended regular expression that matches any word of the form $w_1 \cdots w_n$ with $n \geq 0$, where α matches each w_i with $1 \leq i \leq n$.
6. If α is an extended regular expression that matches a word w, and $x \in X$, then $(\alpha)\%x$ is an extended regular expression that matches the word w, and x is bound to the value w.

We denote the set of all extended regular expressions by RegEx. For every extended regular expression α, we use $L(\alpha)$ to denote the set of all words that are matched by α, and call $L(\alpha)$ *the language generated by* α. A *proper regular expression* is an extended regular expression that contains neither %, nor any variable.

Note that, as in [1], some peculiarities of the semantics of extended regular expressions are not addressed in this definition (some examples are mentioned further down). Câmpeanu et al. [5] offer an alternative definition that explicitly deals with some technical peculiarities that are omitted in Aho's definition, and is closer to the syntax of the programming language PERL. The proofs presented in this paper are not affected by these differences and can be easily adapted to the definition of Câmpeanu et al., or any similar definition, like the models proposed by Bordihn et al. [3], Câmpeanu and Yu [7].

We shall use the notation $(\alpha)^+$ as a shorthand for $\alpha(\alpha)^*$, and freely omit parentheses whenever the meaning remains unambiguous. When doing this, we assume that there is a precedence on the order of the applications of operations, with * and $^+$ ranking over concatenation ranking over the alternation operator |.

We illustrate the intended semantics of extended regular expressions using the following examples in addition to the examples in Section 1:

▶ **Example 4.** Consider the following extended regular expressions:

$$\alpha_1 := \left((\mathtt{a} \mid \mathtt{b})^*\right) \%x\, xx \ \left((\mathtt{a} \mid \mathtt{b})^*\right) \%x\, x, \qquad\qquad \alpha_2 := \left(\left((\mathtt{a} \mid \mathtt{b})^*\right) \%x\, x\right)^+ .$$

These expressions generate the following languages:

$$L(\alpha_1) = \left\{vvvww \mid v, w \in \{\mathtt{a}, \mathtt{b}\}^*\right\},$$
$$L(\alpha_2) = \left\{w_1w_1 \cdots w_nw_n \mid n \geq 1, w_i \in \{\mathtt{a}, \mathtt{b}\}^*\right\}.$$

Note that both expressions rely on the fact that variables can be bound multiple times, and implicitly assume that we parse from left to right. ◇

From a formal point of view, the fact that variables have a scope and the possibility to rebind variables (as in Example 4) can cause unexpected side effects and would normally require a more formal definition of semantics, instead of our "definition by example". Furthermore, Aho's definition does not deal with pathological cases in which some variables might be unbound, e. g., like $((\mathtt{a})\%x \mid \mathtt{b})\, x$. Although the definition by Câmpeanu et al. [5] addresses these problems, we still use Aho's notation, because it is more convenient for the proof of Theorem 19, the main technical tool of the present paper. As mentioned above, all proofs can be easily adapted to the notation and semantics of Câmpeanu et al.

In general, the membership problem for RegEx is NP-complete, as shown in Theorem 6.2 in Aho [1]. As explained in that proof, this problem is solvable in polynomial-time if the number of different variables is bounded. It is not clear how (or if) Aho's reasoning applies to expressions like α_2 in our Example 4; therefore, we formalize a slightly stronger restriction than Aho, and consider the following subclasses of RegEx:

▶ **Definition 5.** For $k \geq 0$, let RegEx(k) denote the class of all extended regular expressions α that satisfy the following properties:

1. α contains at most k occurrences of the metacharacter %,
2. if α contains a subexpression $(\beta)^*$, then the metacharacter % does not occur in β,
3. for every $x \in X$ that occurs in α, α contains exactly one occurrence of $\%x$.

Intuitively, these restrictions on extended regular expressions in RegEx(k) limit not only the number of different variables, but also the total number of possible variable bindings, to at most k.

Note that RegEx(0) is equivalent to the class of proper regular expressions; furthermore, observe that $\mathrm{RegEx}(k) \subset \mathrm{RegEx}(k+1)$ for every $k \geq 0$.

Referring to the extended regular expressions given in Example 4, we observe that, as $\%x$ occurs twice in α_1, α_1 is not element of any RegEx(k) with $k \geq 0$, but the extended regular expression $\alpha_1' := \big((\mathtt{a} \mid \mathtt{b})^*\big)\, \%x\, x x\, \big((\mathtt{a} \mid \mathtt{b})^*\big)\, \%y\, y$ generates the same language as α_1, and $\alpha_1' \in (\mathrm{RegEx}(2) \setminus \mathrm{RegEx}(1))$. In contrast to this, $\alpha_2 \notin \mathrm{RegEx}(k)$ for all $k \geq 0$, as % occurs inside a $()^*$ subexpression (as we defined $^+$ through *).

For any $k \geq 0$, we say that a language L is a RegEx(k)-*language* if there is some $\alpha \in \mathrm{RegEx}(k)$ with $L(\alpha) = L$.

We also consider the class FRegEx of all extended regular expressions that do not use the operator * (or $^+$), and its subclasses $\mathrm{FRegEx}(k) := \mathrm{FRegEx} \cap \mathrm{RegEx}(k)$ for $k \geq 0$. Thus, FRegEx contains exactly those expressions that generate finite (and, hence, regular) languages. Analogously, for every $k \geq 0$, we define a class CoFRegEx(k) as the class of all $\alpha \in \mathrm{RegEx}(k)$ such that $L(\alpha)$ is cofinite. Unlike the classes FRegEx(k), these classes have no straightforward syntactic definition – as we shall prove in Theorem 9, cofiniteness is not semi-decidable for RegEx(k) (if $k \geq 1$).

2.3 Decision Problems and Descriptional Complexity

Most of the technical reasoning in the present paper is centered around the following decision problems:

▶ **Definition 6.** Let Σ denote a fixed terminal alphabet. For all $k, l \geq 0$, we define the following decision problems for RegEx(k):

Universality Given $\alpha \in \mathrm{RegEx}(k)$, is $L(\alpha) = \Sigma^*$?

Cofiniteness Given $\alpha \in \mathrm{RegEx}(k)$, is $\Sigma^* \setminus L(\alpha)$ finite?

***RegEx*(*l*)-ity** Given $\alpha \in \mathrm{RegEx}(k)$, is there a $\beta \in \mathrm{RegEx}(l)$ with $L(\alpha) = L(\beta)$?

As we shall see, Theorem 9 – one of our main technical results – states that these problems are undecidable (to various degrees). We use the undecidability of the universality problem to show that there is no effective procedure that minimizes extended regular expressions with respect to their length, and the undecidability of RegEx(l)-ity to conclude the same for minimization with respect to the number of variables. Furthermore, cofiniteness and RegEx(l)-ity help us to obtain various results on the relative succinctness of proper and extended regular expressions.

By definition, RegEx(l)-ity holds trivially for all RegEx(k) with $k \leq l$. If $l = 0$, we mostly use the more convenient term *regularity* (for RegEx(k)), instead of RegEx(0)-ity. Note that, even for RegEx(0), universality is already PSPACE-complete (see Aho et al. [2]).

In order to examine the relative succinctness of RegEx(1) in comparison to RegEx(0), we use the following notion of complexity measures:

▶ **Definition 7.** Let $\mathcal{R}$ be a class of extended regular expressions. A *complexity measure* for $\mathcal{R}$ is a total recursive function $c : \mathcal{R} \to \mathbb{N}$ such that, for every alphabet Σ, the set of all $\alpha \in \mathcal{R}$ with $L(\alpha) \subseteq \Sigma^*$ **1.** can be effectively enumerated in order of increasing $c(\alpha)$, and **2.** does not contain infinitely many extended regular expressions with the same value $c(\alpha)$.

This definition includes the canonical concept of the length, as well as most of its natural extensions (for example, in our context, one could define a complexity measure that gives additional weight to the number or distance of occurrences of variables, or their nesting level). Kutrib [19] provides more details on (and an extensive motivation of) complexity measures. Using this definition, we are able to define the notion of tradeoffs between classes of extended regular expressions:

▶ **Definition 8.** Let $k > l \geq 0$ and let c be a complexity measure for RegEx(k) (and thereby also for RegEx(l)). A recursive function $f_c : \mathbb{N} \to \mathbb{N}$ is said to be a *recursive upper bound for the tradeoff between* RegEx(k) *and* RegEx(l) if, for all those $\alpha \in$ RegEx(k) for wich $L(\alpha)$ is a RegEx(l)-language, there is a $\beta \in$ RegEx(l) with $L(\beta) = L(\alpha)$ and $c(\beta) \leq f_c(c(\alpha))$.

If no recursive upper bound for the tradeoff between RegEx(k) and RegEx(l) exists, we say that *the tradeoff between* RegEx(k) *and* RegEx(l) *is non-recursive.*

There is a considerable amount of literature on a wide range of non-recursive tradeoffs between various description mechanisms; for a survey, see Kutrib [19].

3 Main Results

As mentioned in Section 1, the central questions of the present paper are whether we can minimize extended regular expressions (under any complexity measure as defined in Definition 7, or with respect to the number variables), and whether there is a recursive upper bound on the tradeoff between extended and proper regular expressions. We approach these questions by proving various degrees of undecidability for the decision problems given in Definition 6, as shown in the main theorem of this section:

▶ **Theorem 9.** *For* RegEx(1)*, universality is not semi-decidable; and regularity and cofiniteness are neither semi-decidable, nor co-semi-decidable.*

The proof of Theorem 9 requires considerable technical preparation and can be found in Section 4.

Of course, all these undecidability results also hold for every RegEx(k) with $k \geq 2$, and for the whole class RegEx of extended regular expressions (as RegEx(1) is contained in all these classes)[2]. Theorem 9 also demonstrates that inclusion and equivalence are undecidable for RegEx(1) (and, hence, all of RegEx). We also see, as an immediate consequence to Theorem 9, that there is no algorithm that minimizes the number of variables in an extended regular expression, as such an algorithm could be used to decide regularity.

Note that in the proof of Theorem 9, the single variable x is bound only to words that match the expression 0^*. This shows that the "negative" properties of extended regular expressions we derive from Theorem 9 hold even if we restrict RegEx(1) by requiring that the variable can only be bound to a very restricted proper regular expression. Furthermore, the proof also applies to the extension of proper regular expressions through numerical parameters that is proposed in Della Penna et al. [12]. In addition to this, the construction from Theorem 19 (which we shall use to prove Theorem 9, and consequently, all other results in the present paper) can be refined to also include bounds on the number of occurrences of the single variable.

From the undecidability of universality, we can immediately conclude that RegEx(1) cannot be minimized effectively:

▶ **Corollary 10.** *Let c be a complexity measure for* RegEx(1)*. Then there is no recursive function m_c that, given an expression $\alpha \in$* RegEx(1)*, returns an expression $m_c(\alpha) \in$* RegEx(1) *with **1.** $L(m_c(\alpha)) = L(\alpha)$, and **2.** $c(\beta) \geq c(m_c(\alpha))$ for every $\beta \in$* RegEx(1) *with $L(\beta) = L(\alpha)$.*

Following the classic proof method of Hartmanis [15] (cf. Kutrib [19]), we can use the fact that non-regularity is not semi-decidable to obtain a result on the relative succinctness of extended and proper regular expressions:

▶ **Corollary 11.** *There are non-recursive tradeoffs between* RegEx(1) *and* RegEx(0)*. This holds even if we consider only the tradeoffs between* CoFRegEx(1) *and* CoFRegEx(0)*, using a complexity measure for* RegEx(1)*.*

Thus, no matter which complexity measure and which computable upper bound we assume for the tradeoff, there is always a regular language L that can be described by an *extended* regular expression from RegEx(1) so much more succinctly that every *proper* regular expression for L has to break that bound. Obviously, this has also implications for the complexity of matching regular expressions: Although membership is "easier" for proper regular expressions than for extended regular expressions, there are regular languages that can be expressed far more efficiently through extended regular expressions than through proper regular expressions.

Recall Example 1, where we consider extended regular expressions that describe finite languages. In this restricted case, there exists an effective conversion procedure – hence, the tradeoffs are recursive:

▶ **Lemma 12.** *For every $k \geq 1$, the tradeoff between* FRegEx(k) *and* FRegEx(0) *is recursive (even when considering complexity measures for* RegEx(k) *instead of* FRegEx(k)*).*

Although the class of RegEx-languages is not closed under complementation (Lemma 2 in Câmpeanu et al. [5]), there are languages L such that both L and its complement $\Sigma^* \setminus L$ are RegEx-languages (e. g., all regular languages). Combining Lemma 12 and Corollary 11, we can straightforwardly conclude that there are cases where it is far more efficient to describe the complement of a RegEx(1)-language, as opposed to the language itself:

[2] Note that cofiniteness for extended regular expressions is a more general case of the question whether a pattern is avoidable over a fixed terminal alphabet, an important open problem in pattern avoidance (cf. Currie [10]). Example 2 illustrates this connection for the pattern xx over a binary alphabet.

▶ **Corollary 13.** *Let Σ be a finite alphabet. Let c be a complexity measure for* $\mathrm{RegEx}(1)$*. For any recursive function $f_c : \mathbb{N} \to \mathbb{N}$, there exists an $\alpha \in \mathrm{RegEx}(1)$ such that $\Sigma^* \setminus L(\alpha)$ is a* $\mathrm{RegEx}(1)$*-language, and for every $\beta \in \mathrm{RegEx}(1)$ with $L(\beta) = \Sigma^* \setminus L(\alpha)$, $c(\beta) \geq f_c(c(\alpha))$.*

With some additional technical effort, we can extend the previous results on undecidability of $\mathrm{RegEx}(l)$-ity and on tradeoffs between $\mathrm{RegEx}(k)$ and $\mathrm{RegEx}(0)$ to arbitrary levels of the hierarchy of $\mathrm{RegEx}(k)$-languages:

▶ **Lemma 14.** *Let $k \geq 1$. For* $\mathrm{RegEx}(k+1)$*,* $\mathrm{RegEx}(k)$*-ity is neither semi-decidable, nor co-semi-decidable.*

In this proof, we concatenate the languages from the proof of Theorem 9 with languages that are $\mathrm{RegEx}(k+1)$-languages, but not $\mathrm{RegEx}(k)$-languages. Non-recursive tradeoffs between $\mathrm{RegEx}(k+1)$ and $\mathrm{RegEx}(k)$ for every $k \geq 1$ follow immediately, using Hartmanis' proof technique as in the proof of Corollary 11.

4 Proof of Theorem 9

On a superficial level, we prove Theorem 9 by using Theorem 19 (which we introduce further down in the present section) to reduce various undecidable decision problems for Turing machines to appropriate problems for extended regular expressions (the problems from Definition 6). This is done by giving an effective procedure that, given a Turing machine $\mathcal{M}$, returns an extended regular expression that generates the complement of a language that encodes all accepting runs of $\mathcal{M}$.

On a less superficial level, this approach needs to deal with certain technical peculiarities that make it preferable to study a variation of the Turing machine model. An *extended Turing machine* is a 3-tuple $\mathcal{X} = (Q, q_1, \delta)$, where Q and q_1 denote the state set and the initial state. All extended Turing machines operate on the tape alphabet $\Gamma{:=}\{0,1\}$ and use 0 as the blank letter. The transition function δ is a function $\delta : \Gamma \times Q \to (\Gamma \times \{L, R\} \times Q) \cup \{\mathrm{HALT}\} \cup (\{\mathrm{CHECK_R}\} \times Q)$. The movement instructions L and R and the HALT-instruction are interpreted canonically – if $\delta(a, q) = (b, M, p)$ for some $M \in \{L, R\}$ (and $a, b \in \Gamma$, $p, q \in Q$), the machine replaces the symbol under the head (a) with b, moves the head to the left if $M = L$ (or to the right if $M = R$), and enters state p. If $\delta(a, q) = \mathrm{HALT}$, the machine halts and accepts.

The command $\mathrm{CHECK_R}$ works as follows: If $\delta(a, q) = (\mathrm{CHECK_R}, p)$ for some $p \in Q$, $\mathcal{X}$ immediately checks (without moving the head) whether the right side of the tape (i. e., the part of the tape that starts immediately to the right of the head) contains only the blank symbol 0. If this is the case, $\mathcal{X}$ enters state p; but if the right side of the tape contains any occurrence of 1, $\mathcal{X}$ stays in q_i. As the tape is never changed during a $\mathrm{CHECK_R}$-instruction, this leads $\mathcal{X}$ into an infinite loop, as it will always read a in q_i, and will neither halt, nor change its state, head symbol, or head position. Although it might seem counterintuitive to include an instruction that allows our machines to search the whole infinite side of a tape in a single step and without moving the head, this command is expressible in the construction we use in the proof of Theorem 19, and it is needed for the intended behavior.

We partition the tape of an extended Turing machine $\mathcal{X}$ into three disjoint areas: The *head symbol*, which is (naturally) the tape symbol at the position of the head, the *right tape side*, which contains the tape word that starts immediately to the right of the head symbol and extends rightward into infinity, and the *left tape side*, which starts immediately left to the head symbol and extends infinitely to the left. When speaking of a configuration, we denote the head symbol by a and refer to the contents of the left or right tape side as the *left* *tape word* t_L or the *right tape word* t_R, respectively. A *configuration* of an extended Turing machine $\mathcal{X} = (Q, q_1, \delta)$ is a tuple (t_L, t_R, a, q), where $t_L, t_R \in \Gamma^* 0^\omega$ are the left and right tape word, $a \in \Gamma$ is the head symbol, and $q \in Q$ denotes the current state. The symbol $\vdash_{\mathcal{X}}$ denotes the successor relation on configurations of $\mathcal{X}$, i. e., $C \vdash_{\mathcal{X}} C'$ if $\mathcal{X}$ enters C' immediately after C.

We define $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$, the *domain* of an extended Turing machine $\mathcal{X} = (Q, q_1, \delta)$, to be the set of all tape words $t_R \in \Gamma^* 0^\omega$ such that $\mathcal{X}$, if started in the configuration $(0^\omega, t_R, 0, q_1)$, halts after finitely many steps.

The definition of $\mathrm{dom}_{\mathrm{X}}$ is motivated by the properties of the encoding that we shall use. Usually, definitions of the domain of a Turing machine rely on the fact that the end of the input is marked by a special letter \$ or an encoding thereof (cf. Minsky [21]). As we shall see, our use of extended regular expressions does not allow us to express the fact that every input is ended by exactly one \$ symbol. Without the $\mathrm{CHECK}_{\mathrm{R}}$-instruction in an extended Turing machine $\mathcal{X}$, we then would have to deal with the unfortunate side effect that a nonempty $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ could never be finite: Assume $w \in \Gamma^*$ such that $w\,0^\omega \in \mathrm{dom}_{\mathrm{X}}(\mathcal{X})$. The machine can only see a finite part of the right side of the tape before accepting. Thus, there is a $v \in \Gamma^*$ such that both $wv1\,0^\omega \in \mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ and $wv0\,0^\omega \in \mathrm{dom}_{\mathrm{X}}(\mathcal{X})$, as $\mathcal{X}$ will not reach the part where $wv1$ and $wv0$ differ. This observation leads to $wvx\,0^\omega \in \mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ for every $x \in \Gamma^*$, and applies to various other extensions of the Turing machine model. As Lemma 18 – and thereby most of the main results in Section 3 – crucially depends on the fact that there are extended Turing machines with a finite domain, we use $\mathrm{CHECK}_{\mathrm{R}}$ to allow our machines to perform additional sanity checks on the input and to overcome the limitations that arise from the lack of the input markers ¢ and \$.

Using a classical coding technique for two-symbol Turing machines (see Minsky [21]) and the correspond undecidability results, we establish the following negative results on decision problems for extended Turing machines:

▶ **Lemma 15.** *Consider the following decision problems for extended Turing machines:*
Emptiness *Given an extended Turing machine $\mathcal{X}$, is $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ empty?*
Finiteness *Given an extended Turing machine $\mathcal{X}$, is $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ finite?*
Then emptiness is not semi-decidable, and finiteness is neither semi-decidable, nor co-semi-decidable.

In order to simplify some technical aspects of our further proofs below, we adopt the following convention on extended Turing machines:

▶ Convention 16. Every extended Turing machine

1. has the state set $Q = \{q_1, \dots, q_\nu\}$ for some $\nu \geq 1$, where q_1 is the initial state,
2. has $\delta(0, q_1) = (0, L, q_2)$,
3. has $\delta(a, q) = \mathrm{HALT}$ for at least one pair $(a, q) \in \Gamma \times Q$.

Obviously, every extended Turing machine can be straightforwardly (and effectively) adapted to satisfy these criteria.

As every tape word contains only finitely many occurrences of 1, we can interpret tape sides as natural numbers in the following (canonical) way: For sequences $t = (t_i)_{i=0}^\infty$ over Γ, define $\mathrm{e}(t) := \sum_{i=0}^\infty 2^i\,\mathrm{e}(t_i)$, where $\mathrm{e}(0) := 0$ and $\mathrm{e}(1) := 1$. Most of the time, we will not distinguish between single letters and their values under e, and simply write a instead of $\mathrm{e}(a)$ for all $a \in \Gamma$. It is easily seen that e is a bijection between $\mathbb{N}$ and $\Gamma^* 0^\omega$, the set of all tape words over Γ. Intuitively, every tape word is read as a binary number, starting with the cell closest to the head as the least significant bit, extending toward infinity.

Expressing the three parts of the tape (left and right tape word and head symbol) as natural numbers allows us to compute the tape parts of successor configurations using elementary integer operations. The following straightforward observation shall be a very important tool in the proof of Theorem 19:

▶ Observation 17. Assume that an extended Turing machine $\mathcal{X} = (Q, q_1, \delta)$ is in some configuration $C = (t_L, t_R, a, q_i)$, and $\delta(a, q_i) = (b, M, q_j)$ for some $b \in \Gamma$, some $M \in \{L, R\}$ and some $q_j \in Q$. For the (uniquely defined) successor configuration $C' = (t'_L, t'_R, a', q_j)$ with $C \vdash_{\mathcal{X}} C'$, the following holds:

$$\begin{array}{llll} \text{If } M = L\text{:} & \mathrm{e}(t'_L) = \mathrm{e}(t_L) \operatorname{div} 2, & \mathrm{e}(t'_R) = 2\,\mathrm{e}(t_R) + b, & a' = \mathrm{e}(t_L) \bmod 2, \\ \text{if } M = R\text{:} & \mathrm{e}(t'_L) = 2\,\mathrm{e}(t_L) + b, & \mathrm{e}(t'_R) = \mathrm{e}(t_R) \operatorname{div} 2, & a' = \mathrm{e}(t_R) \bmod 2. \end{array}$$

These equations are fairly obvious – when moving the head in direction M, $\mathcal{X}$ turns the tape cell that contained the least significant bit of $\mathrm{e}(t_M)$ into the new head symbol, while the other tape side gains the tape cell containing the new letter b that was written over the head symbol as new least significant bit.

Using the encoding e, we define an encoding enc of configurations of $\mathcal{X}$ by

$$\mathrm{enc}\,(t_L, t_R, a, q_i) := 00^{\mathrm{e}(t_L)}\texttt{\#}00^{\mathrm{e}(t_R)}\texttt{\#}00^{\mathrm{e}(a)}\texttt{\#}0^i$$

for every configuration (t_L, t_R, a, q_i) of $\mathcal{X}$. We extend enc to an encoding of finite sequences $C = (C_i)_{i=1}^n$ (where every C_i is a configuration of $\mathcal{X}$) by

$$\mathrm{enc}(C) := \texttt{\#\#}\,\mathrm{enc}(C_1)\texttt{\#\#}\,\mathrm{enc}(C_2)\texttt{\#\#}\cdots\texttt{\#\#}\,\mathrm{enc}(C_n)\texttt{\#\#}.$$

A *valid computation of* $\mathcal{X}$ is a sequence $C = (C_i)_{i=1}^n$ of configurations of $\mathcal{X}$ where C_1 is an initial configuration (i. e. some configuration $(0^\omega, w, 0, q_1)$ with $w \in \Gamma^* 0^\omega$), C_n is a halting configuration, and for every $i < n$, $C_i \vdash_{\mathcal{X}} C_{i+1}$. Thus, let

$$\begin{aligned} \mathrm{VALC}(\mathcal{X}) &= \{\mathrm{enc}(C) \mid C \text{ is a valid computation of } \mathcal{X}\}, \\ \mathrm{INVALC}(\mathcal{X}) &= \{0, \texttt{\#}\}^* \setminus \mathrm{VALC}(\mathcal{X}). \end{aligned}$$

The main part of the proof of Theorem 9 is Theorem 19 (still further down), which states that, given an extended Turing machine $\mathcal{X}$, one can effectively construct an expression from RegEx(1) that generates INVALC($\mathcal{X}$). Note that in enc(C), ## serves as a boundary between the encodings of individual configurations, which will be of use in the proof of Theorem 19. Building on Convention 16, we observe the following fact on the regularity of VALC($\mathcal{X}$) for a given extended Turing machine $\mathcal{X}$:

▶ **Lemma 18.** *For every extended Turing machine* $\mathcal{X}$*,* $\mathrm{VALC}(\mathcal{X})$ *is regular if and only if* $\mathrm{dom}_{\mathrm{X}}(\mathcal{X})$ *is finite.*

The *if* direction is obvious, the *only if* direction follows from Convention 16 and the application of a generalized sequential machine. We are now ready to state the central part of our proof of Theorem 9:

▶ **Theorem 19.** *For every extended Turing machine* $\mathcal{X}$*, one can effectively construct an extended regular expression* $\alpha_{\mathcal{X}} \in \mathrm{RegEx}(1)$ *such that* $L(\alpha_{\mathcal{X}}) = \mathrm{INVALC}(\mathcal{X})$*.*

In the present paper, we only sketch the proof of Theorem 19. Given an extended Turing machine $\mathcal{X}$, the expression $\alpha_{\mathcal{X}}$ can be assembled from various subexpressions that describe a complete list of sufficient criteria for membership in INVALC($\mathcal{X}$). Intuitively, every word in INVALC($\mathcal{X}$) contains (at least) one error. We first consider so-called *structural errors*, where a word is not an encoding of any sequence $(C_i)_{i=1}^n$ over configurations of $\mathcal{X}$ for some n, or the word is such an encoding, but C_1 is not an initial, or C_n is not a halting configuration. These errors can be described by a proper regular expression, which can be straightforwardly derived from the definition of $\mathcal{X}$.

If a word in INVALC($\mathcal{X}$) does not contain any structural errors, it is an encoding of some sequence of configurations $(C_i)_{i=0}^n$ of $\mathcal{X}$, but there is a configuration C_i with $i < n$ such that $C_i \vdash_{\mathcal{X}} C_{i+1}$ does not hold. We call these types of errors *behavioral errors*, and distinguish *state*, *head*, and *tape side errors*, depending on which part of the configuration contains an error. We can describe state and head errors in words that do not contain structural errors using proper regular expressions; the variable is only used in the description of tape side errors. Here, Observation 17 allows us to specify all errors where the e-value of a left or right tape side is too small or too large. Furthermore, all these subexpressions are in RegEx(1), and start with $\#0(0^*)\%x$. This allows us to combine them into a single expression from RegEx(1). Then the expressions for all types of error can be combined to a single expression $\alpha_{\mathcal{X}} \in$ RegEx(1) with $L(\alpha_{\mathcal{X}}) =$ INVALC($\mathcal{X}$), which concludes the proof of Theorem 19.

Theorem 9 follows almost immediately from Theorem 19 and Lemmas 15 and 18. Note that the encoding enc and various parts of the proof of Theorem 19 where inspired by the author's proof of a similar but more narrow result on pattern languages (Bremer and Freydenberger [4]).

5 Conclusions

The present paper shows that extending regular expressions with only a single variable already leads to an immense increase in succinctness and expressive power. The good part of this news is that in certain applications, using the right extended regular expression instead of a proper regular expression can lead to far more efficient running times, even with the same matching engine. The bad part of this news is that this additional power can only be harnessed in full if one is able to solve undecidable problems, which greatly diminishes the usefulness of extended regular expressions as more efficient alternative to proper regular expressions.

Due to underlying undecidable problems, some questions of designing optimal extended regular expressions are of comparable difficulty to designing optimal programs. For applied computer scientists, it could be worthwhile to develop heuristics and good practices to identify cases where the non-conventional use of extended regular expressions might offer unexpected speed advantages. For theoretical computer scientists, the results in the present paper highlight the need for appropriate restrictions other than the number of variables; restrictions that lead to large and natural subclasses with decidable decision problems. One possible approach that does not extend the expressive power of proper regular expressions beyond regular languages would be a restriction of the length of the words on which variables can be bound. As the results in the present paper show, any extension of proper regular expressions that includes some kind of repetition operator needs to be approached with utmost care.

Acknowledgements

The author wishes to thank Nicole Schweikardt and the anonymous referees for their helpful remarks.

References

1 A.V. Aho. Algorithms for finding patterns in strings. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, chapter 5, pages 255–300. Elsevier, 1990.

2 A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*, chapter 10.6. Addison-Wesley, Reading, MA, 1974.

3 H. Bordihn, J. Dassow, and M. Holzer. Extending regular expressions with homomorphic replacements. *RAIRO Theoretical Informatics and Applications*, 44(2):229–255, 2010.

4 J. Bremer and D.D. Freydenberger. Inclusion problems for patterns with a bounded number of variables. In *Proc. DLT 2010*, volume 6224 of *LNCS*, pages 100–111, 2010.

5 C. Câmpeanu, K. Salomaa, and S. Yu. A formal study of practical regular expressions. *International Journal of Foundations of Computer Science*, 14:1007–1018, 2003.

6 C. Câmpeanu and N. Santean. On the intersection of regex languages with regular languages. *Theoretical Computer Science*, 410(24–25):2336–2344, 2009.

7 C. Câmpeanu and S. Yu. Pattern expressions and pattern automata. *Information Processing Letters*, 92(6):267–274, 2004.

8 B. Carle and P. Narendran. On extended regular expressions. In *Proc. LATA 2009*, volume 5457 of *LNCS*, pages 279–289, 2009.

9 J. Cassaigne. Unavoidable patterns. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, chapter 3, pages 111–134. Cambridge University Press, Cambridge, New York, 2002.

10 James Currie. Open problems in pattern avoidance. *American Math. Monthly*, 100(8):790–793, 1993.

11 N. Cutland. *Computability.* Cambridge University Press, 1980.

12 G. Della Penna, B. Intrigila, E. Tronci, and M. Venturini Zilli. Synchronized regular expressions. *Acta Informatica*, 39(1):31–70, 2003.

13 J.E.F. Friedl. *Mastering Regular Expressions.* O'Reilly Media, Sebastopol, CA, 2nd edition, 2002.

14 I. Glaister and J. Shallit. A lower bound technique for the size of nondeterministic finite automata. *Information Processing Letters*, 59(2):75–77, 1996.

15 J. Hartmanis. On Gödel speed-up and succinctness of language representations. *Theoretical Computer Science*, 26(3):335–342, 1983.

16 M. Holzer and M. Kutrib. The complexity of regular(-like) expressions. In *Proc. DLT 2010*, volume 6224 of *LNCS*, pages 16–30, 2010.

17 J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

18 S.C. Kleene. Representation of events in nerve nets and finite automata. In C.E. Shannon; J. McCarthy; W.R. Ashby, editor, *Automata Studies*, pages 3–42. Princeton University Press, 1956.

19 M. Kutrib. The phenomenon of non-recursive trade-offs. *International Journal of Foundations of Computer Science*, 16(5):957–973, 2005.

20 K.S. Larsen. Regular expressions with nested levels of back referencing form a hierarchy. *Information Processing Letters*, 65(4):169–172, 1998.

21 M.L. Minsky. *Computation: Finite and Infinite Machines.* Prentice-Hall, Upper Saddle River, NJ, 1967.

22 D. Reidenbach and M. Schmid. A polynomial time match test for large classes of extended regular expressions. In *Proc. CIAA 2010*, 2010.

New Exact and Approximation Algorithms for the Star Packing Problem in Undirected Graphs

Maxim Babenko[1] and Alexey Gusakov[2]

1 Moscow State University, Yandex
maxim.babenko@gmail.com

2 Moscow State University, Google
agusakov@gmail.com

Abstract

By a *T-star* we mean a complete bipartite graph $K_{1,t}$ for some $t \leq T$. For an undirected graph G, a *T-star packing* is a collection of node-disjoint T-stars in G. For example, we get ordinary matchings for $T = 1$ and packings of paths of length 1 and 2 for $T = 2$. Hereinafter we assume that $T \geq 2$.

Hell and Kirkpatrick devised an ad-hoc augmenting algorithm that finds a T-star packing covering the maximum number of nodes. The latter algorithm also yields a min-max formula.

We show that T-star packings are reducible to network flows, hence the above problem is solvable in $O(m\sqrt{n})$ time (hereinafter n denotes the number of nodes in G, and m — the number of edges).

For the edge-weighted case (in which weights may be assumed positive) finding a maximum T-packing is NP-hard. A novel $\frac{9}{4}\frac{T}{T+1}$-factor approximation algorithm is presented.

For non-negative node weights the problem reduces to a special case of a max-cost flow. We develop a divide-and-conquer approach that solves it in $O(m\sqrt{n}\log n)$ time. The node-weighted problem with arbitrary weights is more difficult. We prove that it is NP-hard for $T \geq 3$ and is solvable in strongly-polynomial time for $T = 2$.

1998 ACM Subject Classification G.2.2 Graph algorithms, G.1.2 Minimax approximation and algorithms

Keywords and phrases graph algorithms, approximation algorithms, generalized matchings, flows, weighted packings.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.519

1 Introduction

1.1 Preliminaries

Recall the classical *maximum matching problem*: given an undirected graph G the goal is to find a collection M (called a *matching*) of node-disjoint edges covering as many nodes as possible. Motivated by this definition, one may consider an arbitrary (possibly infinite) collection of undirected graphs $\mathcal{G}$, called *allowed*, and ask for a collection $\mathcal{M}$ of node-disjoint subgraphs of G (not necessarily spanning) such that every member of $\mathcal{M}$ is isomorphic to some graph in $\mathcal{G}$. Let the *size* of $\mathcal{M}$ be the total number of nodes covered by the elements of $\mathcal{M}$. The generalized matching problem [8] asks for a $\mathcal{G}$-matching of maximum size.

Clearly, the tractability of the generalized problem depends solely on the choice of $\mathcal{G}$. The case when all graphs in $\mathcal{G}$ are bipartite was investigated by Hell and Kirkpatrick [8]. Roughly speaking, in this case the maximum $\mathcal{G}$-matching problem is NP-hard unless $\mathcal{G} =$

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 519–530
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$\{K_{1,1}, \ldots, K_{1,T}\}$ for some $T \geq 1$. (For a precise statement, see [8, Sec. 4].) This is exactly the case we study throughout the paper.

▶ **Definition 1.** A *T-star* is a graph $K_{1,t}$ for some $1 \leq t \leq T$. For an undirected graph G, a *T-star packing* in G is a collection of node-disjoint subgraphs in G (not necessary spanning) that are isomorphic to some T-stars.

Since 1-star packings are just ordinary matchings and are already extensively studied (see, e.g., [14]), we restrict our attention to the case $T \geq 2$.

The max-size T-star packing problem was addressed in [13, 1, 8] and others. An $O(mn)$-time ad-hoc augmenting path algorithm (hereinafter $n := |VG|$, $m := |EG|$) and a min-max formula are known. In [8] it is noted that a faster $O(m\sqrt{n})$-time algorithm can be derived using the blocking augmentation strategy (see [2, 9]), but we are not aware of any publicly available exposition. A more restrictive variant of the problem, where the stars are required to be node-induced subgraphs, is presented in [12]. An extension to node capacities is given in [15].

1.2 Our Contribution

This paper presents an alternative treatment of T-star packings that is based on network flows. In Section 2 we show how the max-size T-star packing problem reduces to finding a max-value flow in a digraph with $O(n)$ nodes and $O(m)$ arcs. This immediately implies an $O(m\sqrt{n})$-time algorithm for the max-size T-star packing problem.

The above reduction serves two purposes. Firstly, it mitigates the need for ad-hoc tricks and fits star packings into a widely studied field of network flows. Secondly, this reduction provides interesting opportunities for attacking other optimization problems that are related to T-star packings.

Let G be an edge-weighted graph and the goal is to find a T-star packing such that the sum of weights of edges belonging to stars is maximum. This problem is NP-hard and in Section 3 we present a $\frac{9}{4}\frac{T}{T+1}$-factor approximation algorithm, which is based on max-cost flows.

Finally let G be a node-weighted graph and the objective function is the sum of weights of nodes covered by stars. This case is studied in Section 4. For non-negative weights, a divide-and-conquer approach yields a nice $O(m\sqrt{n}\log n)$-time algorithm. For general weights, the complexity of the resulting problem depends on T. For $T = 2$, we give a strongly-polynomial algorithm that employs bidirected network flows. If $T \geq 3$, the problem is NP-hard.

2 Reduction to Network Flows

2.1 Auxiliary Digraphs

In this section we explain the core of our approach that relates star packings to network flows. We employ some standard graph-theoretic notation throughout the paper. For an undirected graph G we denote its sets of nodes and edges by VG and EG, respectively. For a directed graph we speak of arcs rather than edges and denote the arc set of G by AG. A similar notation is used for paths, trees, and etc.

For $U \subseteq VG$, the set of arcs entering (respectively leaving) U is denoted by $\delta^{\mathrm{in}}_G(U)$ and $\delta^{\mathrm{out}}_G(U)$. Also, $\gamma_G(U)$ denotes the set of arcs (or edges) with both endpoints in U and $G[U]$ denotes the subgraph of G induced by U, i.e. $G[U] = (U, \gamma_G(U))$. When the (di-)graph is clear from the context, it is omitted from notation. Also for a function $\varphi\colon U \to \mathbb{R}$ and a subset $U' \subseteq U$, let $\varphi(U')$ denote $\sum_{u \in U'} \varphi(u)$.

Let, as earlier, G be an undirected graph and $T \geq 2$ be an integer. Replace each edge in G by a pair of oppositely directed arcs and denote the resulting digraph by $\overrightarrow{G}$. The following definition is crucial:

▶ **Definition 2.** A subset of arcs $F \subseteq A\overrightarrow{G}$ is called *T-feasible* if for each node $v \in VG$ at most T arcs in F leave v and at most one arc in F enters v.

The above T-feasible arc sets are equivalent to T-star packings in the following sense:

▶ **Theorem 3.** *The maximum size of a T-feasible arc set in G is equal to the maximum size of a T-star packing. Moreover, given a T-feasible arc set F one can turn it in linear time into a T-star packing of size at least $|F|$.*

Before presenting the proof of Theorem 3, let us explain how a max-size T-feasible arc set size can be found. To this aim, split each node $v \in V\overrightarrow{G}$ into two copies, say v^1 and v^2. Each arc $(u, v) \in A\overrightarrow{G}$ is transformed into an arc (u^1, v^2). Two auxiliary nodes are added: a *source* s that is connected to every node v^1, $v \in V\overrightarrow{G}$, by arcs (s, v^1), and a *sink* t that is connected to every node v^2, $v \in V\overrightarrow{G}$, by arcs (v^2, t). We endow each arc (s, v^1) with capacity equal to T, each arc (v^2, t) with unit capacity, and the remaining arcs with infinite capacities. The resulting digraph is denoted by H.

We briefly remind the basic terminology and notation on network flows (see, e.g., [5, 18] and [16, Ch. 10]). Let Γ be a digraph with a distinguished source node s and a sink node t. The nodes in $V\Gamma - \{s, t\}$ are called *inner*. Let $u\colon A\Gamma \to \mathbb{Z}_+$ be integer arc capacities.

▶ **Definition 4.** An integer *u-feasible flow* (or just *feasible flow* if capacities are clear from the context) is a function $f\colon A\Gamma \to \mathbb{Z}_+$ such that: (i) $f(a) \leq u(a)$ for each $a \in A\Gamma$; and (ii) $\operatorname{div}_f(v) = 0$ for each inner node v.

Here $\operatorname{div}_f(v) := f(\delta^{\text{out}}(v)) - f(\delta^{\text{in}}(v))$ denotes the *divergence* of f at v. The *value* of f is $\operatorname{val}(f) := \operatorname{div}_f(s)$. A max-value feasible integer flow can be found in strongly polynomial time (see [18] and [16, Ch. 10]).

Let f is a feasible integer flow in H (regarded as a network with a source s, a sink t, and capacities u). Then $f(u^1, v^2) \in \{0, 1\}$ for each $(u, v) \in A\overrightarrow{G}$, since at most one unit of flow may leave v^2. (Hereinafter we abbreviate $f((u, v))$ to $f(u, v)$.) Define

$$F := \left\{ (u, v) \in A\overrightarrow{G} \mid f(u^1, v^2) = 1 \right\}.$$

Then the u-feasibility of f implies the T-feasibility of F. Moreover, this correspondence between u-feasible integer flows f and T-feasible arc sets F is one-to-one.

The augmenting path algorithm of Ford and Fulkerson [5] computes a max-value flow in H in $O(mn)$ time. Applying blocking augmentations [9, 2], the latter bound can be improved to $O(m\sqrt{n})$. (In fact for networks of the above “bipartite” type, one can prove the bound of $O(m\sqrt{\Delta})$. Here $\Delta := \min(\Delta_s, \Delta_t)$, Δ_s is the sum of capacities of arcs leaving s, and Δ_t is the sum of capacities of arcs entering t.)

Therefore by Theorem 3, a maximum T-star packing can be found in $O(m\sqrt{n})$ time. (The *clique compression technique* [4] implies a somewhat better time bound; however, the speedup is only sublogarithmic.)

2.2 Proof of Theorem 3

The proof consists of two parts. For the easy one, let $\mathcal{P}$ be a T-star packing in G. To construct a T-feasible arc set F, take every star $S \in \mathcal{P}$. Let v be its *central* node (i.e. a node of maximum degree) and $u_1, \ldots, u_t$ be its *leafs* (i.e. the remaining nodes). For $S = K_{1,1}$ the notion of a central node is ambiguous but any choice will do. Add arcs $(v, u_1), \ldots, (v, u_t)$ and also (u_1, v) to F. Clearly F is T-feasible and its size coincides with the number of nodes covered by $\mathcal{P}$.

The reverse reduction is more involved. Consider a T-feasible arc set F. Then F decomposes into a collection of node-disjoint weakly connected components. We deal with each of these components separately and construct a T-star packing $\mathcal{P}$ of size at least $|F|$. Let Q be one of the above components. One can easily see that two cases are possible:

Case I: Q forms a directed out-tree $\mathcal{T}$ where each node has at most T children and the arcs are directed towards leafs. The following pruning is applied iteratively to $\mathcal{T}$. Pick an arbitrary leaf u_1 in $\mathcal{T}$ of maximum depth, let v be the parent of u_1 and $u_2, \ldots, u_t$ be the siblings of u_1. Clearly $t \leq T$. Remove nodes $v, u_1, \ldots, u_t$ together with incident arcs from $\mathcal{T}$ and add to $\mathcal{P}$ a copy of $K_{1,t}$, where v is its center and $u_1, \ldots, u_t$ are the leafs. Repeat the process until $\mathcal{T}$ is empty or consists of a single node (the root r). Each time a star covering $t+1$ nodes is added to $\mathcal{P}$, either $t+1$ (if $u \neq r$) or t (if $u = r$) arcs are removed from $\mathcal{T}$. At the end one gets a T-star packing of size at least $|AQ|$ nodes, as required.

Case II: Q consists of a directed cycle Ω and a number (possibly zero) of directed out-trees attached to it (see Fig. 1(a) for an example). Let $g_0, \ldots, g_{l-1}$ be the nodes of Ω (in the order of their appearance on the cycle). For $i = 0, \ldots, l-1$, let $\mathcal{T}_i$ be the directed out-tree rooted at g_i in Q. (If no tree is attached to g_i, then we regard $\mathcal{T}_i$ as consisting solely of its root node g_i.) Each node in the latter trees has at most T children, and the roots of these trees have at most $T-1$ children. We process the trees $\mathcal{T}_0, \ldots, \mathcal{T}_{l-1}$ like in Case I and obtain a partial packing $\mathcal{P}$. Our final task is to modify $\mathcal{P}$ to satisfy the following condition: each node $v \in VQ$ that has an incoming arc in F is covered by a star in $\mathcal{P}$. So far, the above condition is only violated for nodes in Ω that are not covered by $\mathcal{P}$.

Two subcases are possible. First, suppose that all nodes of Ω are not covered. Then one can cover Ω by a collection of node-disjoint (and also disjoint from $\mathcal{P}$) paths of lengths 1 and 2. Adding these paths to $\mathcal{P}$ finishes the job. (Note that this is exactly where we use the condition $T \geq 2$.)

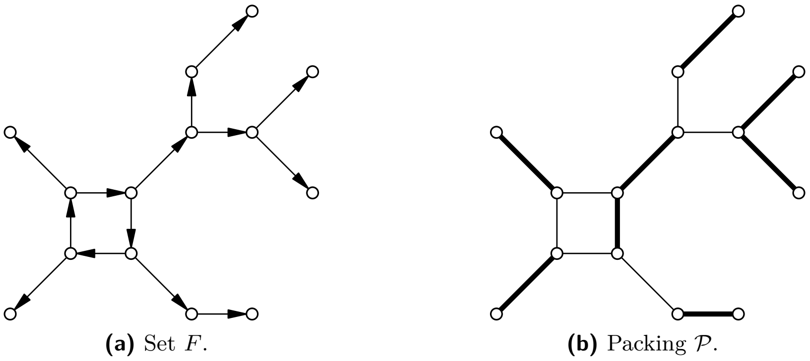
Second, suppose that Ω contains both covered and not covered nodes. Let $g_i, \ldots, g_j$ be a maximal consecutive segment of uncovered nodes, i.e. g_{i-1} and g_{j+1} are covered (indices are taken modulo l). If $j-i$ is odd, then adding $(j-i+1)/2$ disjoint copies of $K_{1,1}$ covering $g_i, \ldots, g_j$ completes the proof. Otherwise let $j-i$ be even. Recall that g_{i-1} is covered by some star $S \in \mathcal{P}$ and g_{i-1} is its central node. Since the degree of g_{i-1} in S is at most $T-1$, one can augment S by adding a new leaf g_i. This way g_i gets covered and the case reduces to the previous one. An example is depicted in Fig. 1(b).

Clearly F can be converted into $\mathcal{P}$ in linear time. ◀

3 Edge-Weighted Packings

3.1 Hardness

Consider arbitrary *edge weights* $w \colon EG \to \mathbb{Q}$ and let the *edge weight* $w(S)$ of a star S be the sum of weights of its edges. In this section we focus on finding a T-star packing $\mathcal{P}$ that maximizes $w(\mathcal{P}) := \sum_{S \in \mathcal{P}} w(S)$. Allowing negative edge weights is redundant since such


(a) Set F.

(b) Packing $\mathcal{P}$.

Figure 1 Transforming F into $\mathcal{P}$ ($T = 2$).

edges may be removed from G without changing the optimum. Therefore we assume that edge weights are non-negative.

▶ **Theorem 5.** *The problem of deciding, for given G, T, w, and $\lambda \in \mathbb{Q}_+$, if G contains a T-star packing of edge weight at least λ, is NP-hard even in the all-unit weight case.*

Proof. It is known (see, e.g. [8]) that deciding if G admits a perfect (i.e. covering all the nodes) $\mathcal{G}$-matching is NP-hard for $\mathcal{G} = \{K_{1,T}\}$. We reduce the latter to the edge-weighted T-star packing problem as follows. If $|VG|$ is not divisible by $|T| + 1$, then the answer is negative. Otherwise set $w(e) := 1$ for all $e \in EG$. A T-star packing $\mathcal{P}$ obeys $w(\mathcal{P}) = \frac{nT}{T+1}$ if and only if all stars in $\mathcal{P}$ are isomorphic to $K_{1,T}$. Hence solving the edge-weighted T-star packing problem enables to check if G has a perfect $\mathcal{G}$-matching. ◀

3.2 Approximation

We show how to compute, in strongly-polynomial time, a T-star packing $\mathcal{P}$ such that $w(\mathcal{P}) \geq \mathrm{OPT} \cdot \frac{4}{9}\frac{T+1}{T}$, where OPT denotes the maximum weight of a T-star packing in G. Let us extend the weights from G to $\overrightarrow{G}$, i.e. define $w(u, v) := w(v, u) := w(e)$ for $e = \{u, v\} \in EG$. Let OPT' be the maximum weight of a T-feasible arc set in $\overrightarrow{G}$.

▶ **Lemma 6.** $\mathrm{OPT}' \geq \mathrm{OPT} \cdot \frac{T+1}{T}$.

Proof. Fix a max-weight packing of T-stars $\mathcal{P}_{\mathrm{OPT}}$. Consider a star $S \in \mathcal{P}_{\mathrm{OPT}}$, and let $e_1 = \{u, v_1\}, \ldots, e_t = \{u, v_t\}$ be the edges forming S ($t \leq T$). We may assume that e_1 is a maximum-weight edge (among $e_1, \ldots, e_t$).

Consider the arc set $\{(u, v_1), (v_1, u), (u, v_2), (u, v_3), \ldots, (u, v_t)\}$ (i.e. e_1 generates a pair of opposite arcs while the other edges — just a single one). Taking the union of all these arc sets one gets a T-feasible arc set F obeying $w(F) \geq \sum_{S \in \mathcal{P}} \frac{T+1}{T} w(S) = \mathrm{OPT} \cdot \frac{T+1}{T}$, as claimed. ◀

Applying the correspondence between feasible integer flows in H and T-feasible arc sets and regarding arc weights as costs, a max-weight T-feasible arc set F can be found by a max-cost flow algorithm in strongly-polynomial time, see [18, Sec. 8.4]. (For arc costs $c\colon AH \to \mathbb{Q}$ and a flow f in H, the *cost* of f is $c(f) := \sum_a c(a) f(a)$.)

We turn F into a T-star packing $\mathcal{P}$ obeying $w(\mathcal{P}) \geq \frac{4}{9} w(F)$ as follows. Consider the weakly-connected components of F and perform a case splitting similar to that in the proof of Theorem 3. For each component Q, we extract a T-star packing $\mathcal{P}_Q$ covering some nodes of Q such that $w(\mathcal{P}_Q) \geq \frac{4}{9} w(Q)$ and then take the union $\mathcal{P} := \bigcup_Q \mathcal{P}_Q$.

Case I: Q is a directed out-tree $\mathcal{T}$ rooted at a node r. Call an arc (u, v) in $\mathcal{T}$ *even* (respectively *odd*) if the length of the r–u path in $\mathcal{T}$ is even (respectively odd). Let E^0 (respectively E^1) denote the set of edges (in G) corresponding to even (respectively odd) arcs of $\mathcal{T}$. Sets E^0 and E^1 generate T-star packings $\mathcal{P}^0$ and $\mathcal{P}^1$ in G. Choose from these a packing with the largest weight and denote it by $\mathcal{P}_Q$. Then $w(\mathcal{P}_Q) \geq \frac{1}{2}\left(w(\mathcal{P}^0) + w(\mathcal{P}^1)\right) = \frac{1}{2} w(Q) \geq \frac{4}{9} w(Q)$.

Case II: Q is a directed cycle Ω with a number of out-trees attached to it. Let $g_0, \ldots, g_{l-1}$ be the nodes of Ω (numbered in the order of their appearance) and $\mathcal{T}_0, \ldots, \mathcal{T}_{l-1}$ be the corresponding trees ($\mathcal{T}_i$ is rooted at g_i, $i = 0, \ldots, l-1$).

Subcase II.1: l is even. Choose an arbitrary node r on Ω and label the arcs of Q as *even* and *odd* as in Case I. (Note that for any node v in Q, there is a unique simple r–v path in Q.) This way, a T-star packing $\mathcal{P}_Q$ obeying $w(\mathcal{P}_Q) \geq \frac{1}{2} w(Q) \geq \frac{4}{9} w(Q)$ is constructed.

Subcase II.2: l is odd. We construct a collection of $3l$ packings (each covering a subset of nodes of Q) of total weight at least $\frac{3l-1}{2} w(Q)$. To this aim, label the arcs of $\mathcal{T}_0, \ldots, \mathcal{T}_{l-1}$ as *even* and *odd* like in Case I (starting from their roots). For $i = 0, \ldots, l-1$, let E_i^0 (respectively E_i^1) be the set of edges (in G) corresponding to even (respectively odd) arcs of $\mathcal{T}_i$. Also let $e_i = \{g_i, g_{i+1}\}$ be the i-th edge of Ω (hereinafter indices are taken modulo l). Consider the (edge sets of the) following l packings (taking $i = 0, \ldots, l-1$):

$$\begin{aligned}
&\{e_i, e_{i+1}\} \cup \{e_{i+3}, e_{i+5}, \ldots, e_{i+l-2}\} \cup \\
&(E_i^1 \cup E_{i+1}^1 \cup E_{i+2}^1) \cup (E_{i+3}^0 \cup E_{i+4}^1) \cup (E_{i+5}^0 \cup E_{i+6}^1) \cup \ldots \cup (E_{i+l-2}^0 \cup E_{i+l-1}^1).
\end{aligned}$$

Also consider the (edge sets of the) following $2l$ packings (taking each value $i = 0, \ldots, l-1$ **twice**):

$$\begin{aligned}
&\{e_{i+1}, e_{i+3}, e_{i+5}, \ldots, e_{i+l-2}\} \cup \\
&E_i^0 \cup (E_{i+1}^0 \cup E_{i+2}^1) \cup (E_{i+3}^0 \cup E_{i+4}^1) \cup \ldots \cup (E_{i+l-2}^0 \cup E_{i+l-1}^1).
\end{aligned}$$

By a straightforward calculation, one can see that the total weight of these $3l$ packings is

$$\begin{aligned}
&\frac{3l-1}{2} \sum_{i=0}^{l} w(e_i) + \frac{3l-1}{2} \sum_{i=0}^{l} w(E_i^0) + \frac{3l+1}{2} \sum_{i=0}^{l} w(E_i^1) \geq \\
&\frac{3l-1}{2} \left(\sum_{i=0}^{l} w(e_i) + \sum_{i=0}^{l} w(E_i^0) + \sum_{i=0}^{l} w(E_i^1) \right) = \frac{3l-1}{2} w(Q).
\end{aligned}$$

Choosing a max-weight packing $\mathcal{P}_Q$ among these $3l$ instances, one gets $w(\mathcal{P}_Q) \geq \frac{1}{3l} \cdot \frac{3l-1}{2} w(Q) \geq \frac{4}{9} w(Q)$ (since $l \geq 3$), as claimed.

The above postprocessing converting F into $\mathcal{P}$ can be done in strongly-polynomial time. Together with Lemma 6 this proves the following:

▶ **Theorem 7.** *A $\frac{9}{4} \frac{T}{T+1}$-factor approximation to the edge-weighted T-star packing problem can be found in strongly polynomial time.*

4 Node-Weighted Packings

4.1 General Weights

Now consider a node-weighted counterpart of the problem. Let $w \colon VG \to \mathbb{Q}$ be node weights, and let the *weight* of a T-star packing $\mathcal{P}$ be the sum of weights of nodes covered by $\mathcal{P}$.

Now one cannot freely assume that weights are non-negative. Indeed, removing a node with a negative weight may change the optimum (consider $G = K_{1,T}$, where the weight of the central node is negative while the weights of the others are positive). In fact, for $T \geq 3$ and arbitrary w, we get an NP-hard problem:

▶ **Theorem 8.** *The problem of deciding, for given G, $T \geq 3$, w, and $\lambda \in \mathbb{Q}$, if G contains a T-star packing of node weight at least λ, is NP-hard.*

Proof. Recall (see [11] and [14, Sec.12.3]) that the following *perfect 3-uniform hypergraph matching problem* is NP-hard: given a nonempty finite domain V, a collection of subsets $\mathcal{E} \subseteq 2^V$, where each element $X \in \mathcal{E}$ is of size 3, and an integer μ, decide if V can be covered by at exactly $\mu := |V|/3$ elements of $\mathcal{E}$.

We reduce this problem to node-weighted 3-star packings as follows. Construct a bipartite graph G taking V as the left part. For each $X = \{v_1, v_2, v_3\} \in \mathcal{E}$ add a node X to the right part and connect it to nodes v_1, v_2, v_3 in the left part. The weights of nodes in the left part are set to M, where M is a sufficiently large positive integer; the weights of nodes in the right part are -1.

Each subcollection $\mathcal{E}' \subseteq \mathcal{E}$ obeying $\bigcup \mathcal{E}' = V$ generates a packing $\mathcal{P}$ of 3-stars (with centers located in the right part and leafs — in the left one). Clearly $w(\mathcal{P}) = M \cdot |V| - |\mathcal{E}'|$.

Vice versa, consider a max-weight packing $\mathcal{P}$ of 3-stars. Assuming $\bigcup \mathcal{E} = V$, $\mathcal{P}$ must cover all nodes in the left part of G (since M is large enough). Let $\mathcal{E}'$ be the set of nodes in the right part of G that are covered by $\mathcal{P}$. Then $\bigcup \mathcal{E}' = V$ and $w(\mathcal{P}) = M \cdot |V| - |\mathcal{E}'|$. Therefore V can be covered by μ elements of $\mathcal{E}$ if and only if G admits a 3-star packing of weight at least $\lambda := M \cdot |V| - \mu$. The reduction is complete. ◀

4.2 Non-Negative Weights

If node weights are non-negative then the problem is tractable. Recall the construction of the auxiliary network H and assign non-negative arc costs $c\colon AH \to \mathbb{Q}$ as follows: $c(v^2, t) := w(v)$ for all $v \in VG$ and $c(a) := 0$ for the other arcs a. Then by Theorem 3 computing a max-cost flow in H also solves the maximum weight T-star packing problem. The max-cost flow problem is solvable in strongly-polynomial time (see [6, 7] and also [16, Ch.12] for a survey) but using a general method here is an overkill. Note that the costs are non-zero only on arcs incident to the sink. This makes the problem essentially lexicographic.

In what follows, we employ an equivalent treatment, which involves *multi-terminal* networks. Namely, let Γ be a digraph endowed with arbitrary arc capacities u. Consider a set of *sources* S and a sink t ($S \subseteq V\Gamma$, $t \in V\Gamma$, $t \notin S$). Nodes in $V\Gamma - S - \{t\}$ are called *inner*. The notion of feasible flows (see Definition 4) extends to multi-terminal networks. Sometimes we use the term *S–t flow* to emphasize that f is a multi-source flow.

The *value* of an S–t flow f is $\operatorname{val}(f) := \sum_{s \in S} \operatorname{div}_f(s)$. Also let $w\colon S \to Q_+$ be *weights* of sources. The *weight* of f is defined as $w(f) := \sum_{s \in S} w(s) \operatorname{div}_f(s)$. The goal is to find a feasible S–t flow f of maximum weight $w(f)$. When $S = \{s\}$ and $w(s) = 1$, this coincides with the usual max-value flow problem.

Clearly this problem is equivalent to its *multi-sink* counterpart (where weights are assigned to sinks rather than sources). Consider the digraph H constructed in Section 2. Splitting the sink t into n copies (one for each node in VG) and assigning weights to these new sinks appropriately, one reduces the node-weighted star packing problem to the max-weight multi-sink flow problem.

In what follows, we deal with the max-weight multi-source flow problem in Γ. To solve the latter, we present a divide-and-conquer algorithm, which is inspired by [17]. Our flow-based approach, however, is more general and is also much simpler to explain.

For $S', T' \subseteq V\Gamma$, $S' \cap T' = \emptyset$, a subset $X \subseteq V\Gamma$ such that $S' \subseteq X$, $T' \cap X = \emptyset$, is called an *S'–T' cut*. When S' or T' is singleton the notation is abbreviated accordingly. A cut X is called *minimum* (among all S'–T' cuts) if $c(\delta^{\text{out}}(X))$ is minimum. A u-feasible flow f is said to *saturate* X if $f(a) = u(a)$ for all $a \in \delta^{\text{out}}(X)$ and $f(a) = 0$ for all $a \in \delta^{\text{in}}(X)$. In other words, $f(\delta^{\text{out}}(X)) = u(\delta^{\text{out}}(X))$ and $f(\delta^{\text{in}}(X)) = 0$.

Recall that for a u-feasible flow f in a digraph Γ, the *residual graph* $\Gamma_f = (V\Gamma_f := V\Gamma, A\Gamma_f)$ contains *forward* arcs $a = (u, v) \in A\Gamma$, where $f(a) < u(a)$ (endowed with the *residual* capacity $u_f(a) := u(a) - f(a)$), and also *backward* arcs $a^{-1} = (v, u)$, where $a = (u, v) \in A\Gamma$, $f(a) > 0$ (endowed with the *residual* capacity $u_f(a^{-1}) := f(a)$). For a u-feasible flow f is Γ and a u_f-feasible flow g in Γ_f the *sum* $f \oplus g$ is a u-feasible flow in Γ defined by $(f \oplus g)(a) := f(a) + g(a) - g(a^{-1})$ (where terms corresponding to non-existent arcs are assumed to be zero).

W.l.o.g. no arc enters a source and no arc leaves a sink in Γ. Sort the sources in the order of decreasing weight: $w(s_1) \geq w(s_2) \geq \ldots \geq w(s_k)$. For $i = 1, \ldots, k$, define $S_i := \{s_1, \ldots, s_i\}$. We find a feasible S–t flow f and a collection of cuts $X_1, \ldots, X_k$ such that:

(1) (i) $X_1 \subseteq X_2 \subseteq \ldots \subseteq X_k$;
(ii) for $i = 1, \ldots, k$, $X_i \cap S = S_i$, $t \notin X_i$, and f saturates X_i.

▶ **Lemma 9.** *If (1) holds, then f is both a max-weight and a max-value flow.*

Proof. Let $d_i := w(s_i) - w(s_{i+1})$ for $i = 1, \ldots, k-1$ and $d_k := w(s_k)$. For $i = 1, \ldots, k$, define $v_i := \text{div}_f(s_1) + \ldots + \text{div}_f(s_i)$. Applying Abel transformation, one gets $w(f) = d_1 v_1 + \ldots d_k v_k$.

Fix $i = 1, \ldots, k$ and describe f as a sum $f' + f''$, where f' is a feasible $\{s_1, \ldots, s_i\}$–t flow and f'' is a feasible $\{s_{i+1}, \ldots, s_k\}$–t flow (such f', f'' exist due to flow decomposition theorems, see [5]). Clearly $\text{val}(f') = v_i$, therefore $v_i \leq c(\delta^{\text{out}}(X_i))$. Summing over $i = 1, \ldots, k$, we get $w(f) \leq d_1 c(\delta^{\text{out}}(X_1)) + \ldots + d_k c(\delta^{\text{out}}(X_k))$. By (1)(ii), the above inequality holds with equality, hence f is a max-weight flow. Also taking $i = k$ in (1)(ii), we see that X_k is an S–t cut saturated by f. Therefore f is a max-value flow. ◀

It remains to explain how one can find f and X_i obeying (1). Consider an instance $I = (\Gamma, S = \{s_1, \ldots, s_k\}, t)$ (the capacities u and the weights w remain fixed during the whole computation and are omitted from notation). If $k = 1$, then solving I reduces to finding a max-value s_1–t flow f and a minimum s_1–t cut X_1.

Otherwise define $l := \lfloor k/2 \rfloor$, $S^1 := \{s_1, \ldots, s_l\}$, and $S^2 := \{s_{l+1}, s_{l+2}, \ldots, s_k\}$. Compute a max-value S^1–t flow h and the corresponding minimum S^1–t cut Z, which is saturated by h. Since no arc enters a source, we may assume that $Z \cap S = S^1$. To proceed with recursion, construct a pair of problem instances as follows. First, contract $\overline{Z} := V\Gamma - Z$ in Γ into a new sink t^1 and denote the resulting instance by $I^1 := (\Gamma^1 := \Gamma/\overline{Z}, S^1, t^1)$. Second, remove the subset Z in Γ_h (together with the incident arcs) and denote the resulting instance by $I^2 := (\Gamma^2 := \Gamma_h - Z, S^2, t)$.

Let f^1 and f^2 be optimal solutions to I^1 and I^2, respectively, which are found recursively and satisfy (1) (for $f := f^1$, $S := S^1$ and for $f := f^2$, $S := S^2$). Construct an optimal solution to I as follows. First, Z is a minimum S^1–t^1 cut in Γ^1 (since Z is a minimum S^1–t cut in Γ) and by Lemma 9, f^1 is a max-value flow. Hence f^1 saturates Z. Second, f^2 may be regarded as an S^2–t flow in Γ_h. The sum $h \oplus f^2$ forms a u-feasible S–t flow in Γ that also saturates Z. "Glue" f^1 and $h \oplus f^2$ along $\delta^{\text{in}}(Z)$, $\delta^{\text{out}}(Z)$ and construct an S–t flow f in Γ as follows:

$$f(a) := \begin{cases} f^1(a) & \text{for } a \in \gamma(Z), \\ (h \oplus f^2)(a) & \text{for } a \in \gamma(\overline{Z}), \\ u(a) & \text{for } a \in \delta^{\text{out}}(Z), \\ 0 & \text{for } a \in \delta^{\text{in}}(Z). \end{cases}$$

Let $X_1^1, X_2^1, \ldots, X_l^1$ and $X_{l+1}^2, X_{l+2}^2, \ldots, X_k^2$ be the sequence of nested cuts (as in (1)) for f^1 and f^2 (respectively). Then clearly $X_1^1, X_2^1, \ldots, X_l^1, Z \cup X_{l+1}^2, Z \cup X_{l+2}^2, \ldots, Z \cup X_k^2$ and f obey (1). The description of the algorithm is complete.

Let $\Phi(n', m')$ denote the complexity of a max-flow computation in a network with n' nodes and m' arcs. Let the above recursive algorithm be applied to a network with n nodes, m arcs, and k sources. Then its running time $T(n, m, k)$ obeys the recurrence

$$T(n, m, k) = \Phi(n, m) + T(n^1, m^1, \lfloor k/2 \rfloor) + T(n^2, m^2, \lceil k/2 \rceil) + O(n + m),$$

where $n^1 + n^2 = n + 1$, $m^1 + m^2 = m$. For a "natural" time bound Φ this yields $T(n, m, k) = O(\Phi(n, m) \cdot \log k)$ (see [10, Sec. 2.3]).

▶ **Theorem 10.** *In a network with n nodes, m arcs, and k sources a max-weight flow can be found in $O(\Phi(n, m) \cdot \log k)$ time.*

For node-weighted star packings, $\Phi(n, m) = O(m\sqrt{n})$ for the max-flow problems arising during the recursive process (due to results of [2, 9]).

▶ **Corollary 11.** *The node-weighted T-star packing problem with non-negative weights is solvable in $O(m\sqrt{n} \log n)$ time.*

4.3 Node-Weighted Packings of 2-Stars

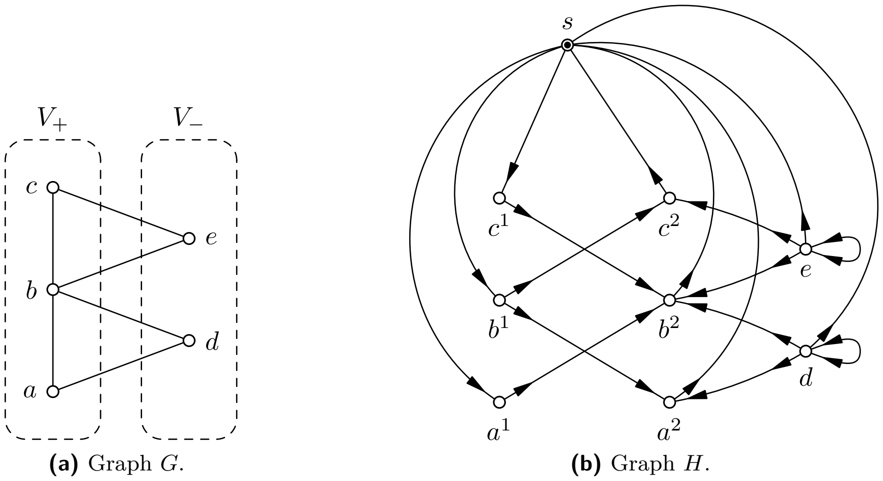
We still have a case where neither a polynomial algorithm nor a hardness result are established. Let $T = 2$ and node weights be arbitrary. Hence T-stars are just paths of length 1 and 2. This case is tractable but the needed machinery is of a bit different nature.

Recall the proof of Theorem 8. The latter fails for $T = 2$ because it shows a reduction from a version of the set cover problem where all subsets are restricted to be of size 1 and 2. The latter set cover problem is equivalent to finding a minimum cardinality *edge cover* in a general (i.e. not necessarily bipartite) graph. Both cardinality and weighted problems regarding edge covers are polynomially solvable (see [16, Ch.27]), so no hardness result can be obtained this way. However, this gives a clue on what techniques may apply here.

We employ the concept of bidirected graphs, which was introduced by Edmonds and Johnson [3] (more about bidirected graphs can be found in, e.g., [16, Ch. 36].) Recall that in a *bidirected* graph edges of three types are allowed: a usual directed edge, or an *arc*, that leaves one node and enters another one; an edge directed *from both* of its ends; and an edge directed *to both* of its ends. When both ends of an edge coincide, the edge becomes a loop.

The notion of a flow is extended to bidirected graphs in a natural fashion. Namely, let Γ is a bidirected graph whose edges are endowed with integer capacities $u \colon E\Gamma \to \mathbb{Z}_+$ and let s be a distinguished node (a *terminal*). Nodes in $V\Gamma - \{s\}$ are called *inner*.

▶ **Definition 12.** A u-feasible (or just feasible) integer bidirected flow f is a function $f \colon E\Gamma \to \mathbb{Z}_+$ such that: (i) $f(e) \le u(e)$ for each $e \in E\Gamma$; and (ii) $\text{div}_f(v) = 0$ for each inner node v.



(a) Graph G. **(b)** Graph H.

Figure 2 Reduction to a bidirected graph.

Here, as usual, $\mathrm{div}_f(v) := f(\delta^{\mathrm{out}}(v)) - f(\delta^{\mathrm{in}}(v))$, where $\delta^{\mathrm{in}}(v)$ denotes the set of edges entering v and $\delta^{\mathrm{out}}(v)$ denotes the set of edges leaving v. It is important to note that a loop e entering (respectively leaving) a node v is counted **two times** in $\delta^{\mathrm{in}}(v)$ (respectively in $\delta^{\mathrm{out}}(v)$) and hence contributes $\pm 2f(e)$ to $\mathrm{div}_f(v)$. Similar to flows in digraphs, $f(\{u, v\})$ is abbreviated to $f(u, v)$.

Consider an undirected graph G endowed with arbitrary node weights $w\colon VG \to \mathbb{Q}$. We reduce the node-weighed 2-star packing problem in G to finding a feasible max-cost integer bidirected flow in an auxiliary bidirected graph. The latter is solvable in strongly polynomial time [16, Ch. 36].

To construct the desired bidirected graph H, denote $V_+ := \{v \in VG \mid w(v) \geq 0\}$ and $V_- := VG \setminus V_+$, Like in Section 2, consider two disjoint copies of V_+ and denote them by V_+^1 and V_+^2. Also add a terminal s and define $VH := V_+^1 \cup V_+^2 \cup V_- \cup \{s\}$.

One may assume that no two nodes in V_- are connected by an edge since these edges may be removed without changing the optimum. For an edge $\{u, v\} \in EG$, $u, v \in V_+$, construct edges $\{u^1, v^2\}$ (leaving u^1 and entering v^2) and $\{v^1, u^2\}$ (leaving v^1 and entering u^2). For an edge $\{u, v\} \in EG$, $u \in V_-$, $v \in V_+$, construct an edge $\{u, v^2\}$ (leaving u^1 and entering v^2). All these bidirected edges are endowed with infinite capacities and zero costs.

For each node $v \in V_+$, add an edge $\{s, v^1\}$ (entering v^1) of capacity 2 and zero cost, and an edge $\{v^2, s\}$ (leaving v^2) of capacity 1 and cost $w(v)$. For each node $v \in V_+$, add a loop $\{v, v\}$ (entering v twice) of capacity 1 and cost $w(v)$ and an edge $\{v, s\}$ (leaving v) of infinite capacity and zero cost. (Since s is a terminal, directions of edges at s are irrelevant.) An example is depicted in Fig. 2.

▶ **Theorem 13.** *The maximum cost of a feasible integer bidirected flow in H coincides with the maximum weight of a 2-star packing in G.*

Proof. We first show how to turn a max-weight 2-star packing $\mathcal{P}$ in G into a feasible integer bidirected flow f in H of cost $w(\mathcal{P})$. Start with $f := 0$. Let S be a star in $\mathcal{P}$. The following cases are possible.

Case I: S covers two nodes, say p and q, and $\{p, q\}$ is the edge of S.

Subcase I.1: $p, q \in V_+$. Increase f by one along the paths (s, p^1, q^2, s) and (s, q^1, p^2, s). This preserves zero divergences at inner nodes and adds $w(p) + w(q) = w(S)$ to $c(f)$.

Subcase I.2: $p \in V_+$, $q \in V_-$. Increase f by one along the path (s, p^2, q, q, s) (where the q, q fragment denotes the loop at q). Divergences at inner nodes are preserved, $c(f)$ is increased by $w(p) + w(q) = w(S)$.

Case II: S covers three nodes, say p, q, and r, and $\{p, q\}, \{q, r\}$ are the edges of S.

Subcase II.1: $p, q, r \in V_+$. Increase f by one along the paths (s, q^1, p^2, s), (s, q^1, r^2, s), and (s, p^1, q^2, s). Divergences at inner nodes are preserved, $c(f)$ is increased by $w(p)+w(q)+w(r) = w(S)$.

Subcase II.2: $p, r \in V_+$ and $q \in V_-$. Increase f by one along the path (s, p^2, q, q, r^2, s) (as above, the q, q fragment is the loop at q). Divergences at inner nodes are preserved, $c(f)$ is increased by $w(p) + w(q) + w(r) = w(S)$.

Since $\mathcal{P}$ is optimal, the other cases are impossible. Applying the above to all $S \in \mathcal{P}$ one gets a feasible integer bidirected flow of cost $w(\mathcal{P})$, as claimed.

For the opposite direction, consider a feasible max-cost integer bidirected flow f in H and construct a 2-star packing $\mathcal{P}$ obeying $w(\mathcal{P}) \ge c(f)$ as follows. Define

$$F_+ := \left\{(u, v) \mid u, v \in V_+,\ f(u^1, v^2) > 0\right\},$$
$$F_- := \left\{(u, v) \mid u \in V_-,\ v \in V_+,\ f(u, v^2) > 0\right\}.$$

Then $F := F_+ \cup F_-$ is a 2-feasible arc set in $\overrightarrow{G}$. (Recall that $\overrightarrow{G}$ is obtained from G by replacing each edge with a pair of opposite arcs.) Indeed, every arc in F leaving a node $u \in V_+$ corresponds to a unit of flow along the edge $\left\{s, u^1\right\}$ and the capacity of the latter is 2. Every arc in F leaving a node $u \in V_-$ corresponds to a unit of flow along the edge $\left\{u, v^2\right\}$, $v \in V_+$, and since the capacity of the loop $\{v, v,\}$ is 1, there can be at most 2 such arcs. Next, if an arc in F enters a node $v \in V_+$ then this arc adds a unit of flow along the edge $\left\{v^2, s\right\}$ (whose capacity is 1). Finally, no arc in F enters a node in V_-.

By Theorem 3, F generates a packing of 2-stars $\mathcal{P}$ in G. We claim that $w(\mathcal{P}) \ge c(f)$. We show that each edge $e \in EH$ with $c(e) > 0$ and $f(e) = 1$ corresponds to a node $v_e \in VG$ covered by $\mathcal{P}$ such that $c(e) = w(v_e)$. Also each node $v \in V_-$ covered by $\mathcal{P}$ corresponds to an edge $e_v \in EH$ with $f(e_v) = 1$ such that $c(e_v) = w(v)$. (The mappings $e \mapsto v_e$ and $v \mapsto e_v$ are injective.) These observations complete the proof of Theorem 13.

For the first part, consider an edge $e = \left\{v^2, s\right\}$, where $f(e) = 1$ and $v \in V_+$. Then v is entered by an arc in F, hence $\mathcal{P}$ covers $v_e := v$. For the second part, consider a node $v \in V_-$ covered by $\mathcal{P}$. Then v must be an endpoint of an arc $a \in F$. No arc in F can enter v (by the construction of F), hence $a = (v, u)$ for $u \in V_+$. Therefore $a \in F_-$ corresponds to the edge $\left\{v, u^2\right\}$. Since $f(v, u^2) > 0$ one has $f(e_v) = 1$, where $e_v := \{v, v\}$ is the loop at v. ◀

Acknowledgements

We thank anonymous referees for useful suggestions.

References

1 A. Amahashi and M. Kano. On factors with given components. *Discrete Math.*, 42(1):1–6, 1982.

2 E. Dinic. Algorithm for solution of a problem of maximum flow in networks with power estimation. *Soviet Math. Dokl.*, 11:1277–1280, 1970.

3 J. Edmonds and E. L. Johnson. Matching, a well-solved class of integer linear programs. In *Proc. Calgary Int. Conf. on Comb. Structures and Their Appl.*, pages 89–92, NY, 1970. Gordon and Breach.

4 T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *J. Comput. Syst. Sci.*, 51:261–272, October 1995.

5 L. Ford and D. Fulkerson. *Flows in Networks.* Princeton University Press, 1962.

6 A. Goldberg and R. Tarjan. Solving minimum-cost flow problems by successive approximation. In *Proc. 18th Annual ACM Conference on Theory of Computing*, pages 7–18, 1987.

7 A. Goldberg and R. Tarjan. Finding minimum-cost circulations by canceling negative cycles. *J. ACM*, 36(4):873–886, 1989.

8 P. Hell and D. Kirkpatrick. Packings by complete bipartite graphs. *SIAM J. Algebraic Discrete Methods*, 7(2):199–209, 1986.

9 J. Hopcroft and R. Karp. An $n^{\frac{5}{2}}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2(4):225–231, 1973.

10 T. Ibaraki, A. Karzanov, and H. Nagamochi. A fast algorithm for finding a maximum free multiflow in an inner eulerian network and some generalizations. *Combinatorica*, 18(1):61–83, 1998.

11 R. Karp. Reducibility among combinatorial problems. In R. Miller and J. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

12 A. Kelmans. Optimal packing of induced stars in a graph. *Discrete Math.*, 173(1-3):97–127, 1997.

13 M. Las Vergnas. An extension of Tutte's 1-factor theorem. *Discrete Math.*, 23:241–255, 1978.

14 L. Lovász and M. D. Plummer. *Matching Theory.* North-Holland, NY, 1986.

15 Q. Ning. On the star packing problem. In *Proc. 1st China-USA International Graph Theory Conference*, volume 576, pages 411–416, 1989.

16 A. Schrijver. *Combinatorial Optimization.* Springer, Berlin, 2003.

17 T. Spencer and E. Mayr. Node weighted matching. In *Proc. 11th Colloquium on Automata, Languages and Programming*, pages 454–464, London, UK, 1984. Springer-Verlag.

18 R. Tarjan. *Data structures and network algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1983.

Balanced Interval Coloring

Antonios Antoniadis, Falk Hüffner, Pascal Lenzner, Carsten Moldenhauer, and Alexander Souza

Institut für Informatik, Humboldt-Universität zu Berlin, D-10099 Berlin, Germany
{antoniad,hueffner,lenzner,moldenha,souza}@informatik.hu-berlin.de

Abstract

We consider the discrepancy problem of coloring n intervals with k colors such that at each point on the line, the maximal difference between the number of intervals of any two colors is minimal. Somewhat surprisingly, a coloring with maximal difference at most one always exists. Furthermore, we give an algorithm with running time $O(n \log n + kn \log k)$ for its construction. This is in particular interesting because many known results for discrepancy problems are non-constructive. This problem naturally models a load balancing scenario, where n tasks with given start- and endtimes have to be distributed among k servers. Our results imply that this can be done ideally balanced.

When generalizing to d-dimensional boxes (instead of intervals), a solution with difference at most one is not always possible. We show that for any $d \geq 2$ and any $k \geq 2$ it is NP-complete to decide if such a solution exists, which implies also NP-hardness of the respective minimization problem.

In an online scenario, where intervals arrive over time and the color has to be decided upon arrival, the maximal difference in the size of color classes can become arbitrarily high for any online algorithm.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems—Sequencing and scheduling

Keywords and phrases Load balancing, discrepancy theory, NP-hardness

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.531

1 Introduction

In this paper, we consider the following load balancing problem: We are given a set $\mathcal{I} = \{I_1, \ldots, I_n\}$ of tasks, where each task is represented by an interval $I = [\ell, r] \in \mathcal{I}$ with starttime ℓ and endtime r. Furthermore, we are given k servers and have to assign the tasks to the servers as evenly as possible. That is, we want to minimize the maximal difference of the numbers of tasks processed by any two servers over all times.

We formalize this in terms of an interval coloring problem: We are given a set $\mathcal{I} = \{I_1, \ldots, I_n\}$ of n intervals on the real line and a set $K = \{1, \ldots, k\}$ of k colors. A k-*coloring* is a mapping $\chi : \mathcal{I} \to K$. For a fixed k-coloring χ and a point $x \in \mathbb{R}$, let $c_i(x)$ denote the number of intervals containing x that have color i in χ. Define the *imbalance* of χ at x by $\operatorname{imb}(x) = \max_{i,j \in K} |c_i(x) - c_j(x)|$. In words, this is the maximum difference in the size of color classes at point x. The *imbalance* of χ is given by $\operatorname{imb}(\chi) = \max_{x \in \mathbb{R}} \operatorname{imb}(x)$.

These definitions yield the following minimization problem:

Minimum Imbalance Interval k-Coloring
Instance: A set of intervals $\mathcal{I}$.
Task: Find a k-coloring χ with minimal $\operatorname{imb}(\chi)$.

We call a k-coloring with imbalance at most one *balanced*. Observe that if the number of intervals intersecting at some point is not divisible by k, then imbalance at least one is unavoidable. On the other hand, if the number of intersecting intervals is divisible by k, then no coloring having imbalance one exists. Thus, if a balanced coloring exists, its imbalance is minimal.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 531–542
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

As we will see shortly, it is always possible to find a balanced interval k-coloring. Hence, we will mostly be concerned with its construction. More specifically, the questions considered in this paper are outlined as follows:

(i) Is there always a balanced k-coloring?
(ii) If so, is it possible to construct a balanced k-coloring in polynomial time?
(iii) If we consider arcs of a circle (instead of intervals), do balanced k-colorings always exist?
(iv) How is the situation if intervals arrive online?
(v) If d-dimensional boxes (instead of intervals) are considered, can the existence of a balanced k-coloring be decided in polynomial time?

The problem has close connections to discrepancy theory; see Doerr [9] and Matoušek [17] for introductions to the field. Let $H = (X, U)$ be a hypergraph consisting of a set X of vertices and a set $U \subseteq 2^X$ of hyperedges. Analogous to the previous definitions, a k-coloring is a mapping $\chi : X \to K$, and the imbalance $\mathrm{imb}(\chi)$ is the largest difference in size between two color classes over all hyperedges. The *discrepancy* problem is to determine the smallest possible imbalance, i. e., $\mathrm{disc}(H) = \min_{\chi:X\to K} \mathrm{imb}(\chi)$.

Hence our problem is to find the discrepancy of the hypergraph $H = (I, U)$, where U is the family of all maximal subsets of intervals intersecting at some point. It turns out that this hypergraph has totally unimodular incidence matrix, which is useful because de Werra [22] proved that balanced k-colorings exist for hypergraphs with totally unimodular incidence matrix. However, the proof in [22] is only partially constructive: A balanced k-coloring is constructed by iteratively solving the problem of balanced 2-coloring on hypergraphs with totally unimodular incidence matrix, for which no algorithm was given in [22].

Further related work in discrepancy theory mostly considers hypergraph coloring with two colors and often from existential, rather than algorithmic perspective. For an arbitrary hypergraph H with n vertices and m hyperedges, the bound $\mathrm{disc}(H) \leq \sqrt{2n \ln(2m)}$ for 2-coloring follows with the probabilistic method; see also [9]. For $m \geq n$, Spencer [20] proved the stronger result $\mathrm{disc}(H) = O(\sqrt{n \log(m/n)})$, which is in particular interesting for $m = O(n)$. If each vertex is contained in at most t edges, the 2-coloring bound $\mathrm{disc}(H) = O(\sqrt{t} \log n)$ was shown by Srinivasan [21] and the bound $\mathrm{disc}(H) \leq 2t - 1$ by Beck and Fiala [5]. Biedl et al. [6] improved the bound to $\mathrm{disc}(H) \leq \max\{2t - 3, 2\}$ for 2-colorings and established $\mathrm{disc}(H) \leq 4t - 3$ for general k-colorings. They also showed that it is NP-complete to decide the existence of balanced k-colorings for hypergraphs with $t \geq \max\{3, k - 1\}$ and $k \geq 2$.

Bansal [4] recently gave efficient algorithms that achieve 2-color imbalances similar to [20, 21] up to constant factors. In particular, an algorithm yields $\mathrm{disc}(H) = O(\sqrt{n} \log(2m/n))$ matching the result of Spencer [20] if $m = O(n)$. Furthermore, $\mathrm{disc}(H) = O(\sqrt{t} \log n)$ complies with the non-constructive result of Srinivasan [21]. For general $k > 2$, Doerr and Srivastav [10] gave a recursive method constructing k-colorings from (approximative) 2-colorings.

Unfortunately, these results on general discrepancy theory do not answer any of the problems considered here, because t is only bounded by the number of vertices.

Our Contributions. We contribute the following answers to the above questions:

(i) Balanced k-colorings exist for any set $\mathcal{I}$ of intervals, i. e., question (i) can *always* be answered in the affirmative. We establish this by showing that our hypergraph H has totally unimodular incidence matrix and then applying a result of de Werra [22]. This also follows independently from our algorithmic results below.

(ii) We present an $O(n \log n)$ time algorithm for finding a balanced 2-coloring, thereby establishing the first constructive result for intervals. Furthermore, we give an $O(n \log n + kn \log k)$ algorithm for finding a balanced k-coloring. This is an improvement in time complexity, since the construction of de Werra [22] combined with our algorithm for 2-coloring only yields $O(n \log n + k^2 n)$. We also note that our algorithm works for any hypergraph with incidence matrix having the consecutive-ones property.

(iii) If we consider arcs of a circle instead of intervals, balanced k-colorings do not exist in general. However, we give an algorithm achieving imbalance at most two with the same time complexity as in the interval case.

(iv) In an online scenario, in which we learn intervals over time, the imbalance of *any* online algorithm can be made arbitrarily high.

(v) For d-dimensional boxes, it is NP-complete to decide if a balanced k-coloring exists for any $d \geq 2$ and any $k \geq 2$. Our reduction is from NOT-ALL-EQUAL 3SAT. This result clearly implies NP-hardness of the respective minimization problem.

2 Interval Colorings

In this section, we consider MINIMUM IMBALANCE INTERVAL k-COLORING, establish the existence of balanced k-colorings, and give algorithms for 2 and k colors, respectively. Later, we consider arcs of a circle and an online version.

2.1 Existence of Balanced k-Colorings

We begin by observing the existence of balanced k-colorings. In the proof below, we use a theorem of de Werra [22], but the existence of balanced k-colorings also follows from our algorithmic results.

▶ **Theorem 1.** *For any set $\mathcal{I}$ of intervals and any $k \in \mathbb{N}$, there is a balanced k-coloring.*

Proof. Let $\mathcal{I}$ be the set of given intervals. Define a hypergraph $H = (I, U)$, where U is the family of all maximal subsets of intervals intersecting at some point. For H with $I = \{I_1, \ldots, I_n\}$ and $U = \{U_1, \ldots, U_m\}$, the incidence matrix is defined by $A = (a_{i,j})$ with $a_{i,j} = 1$ if $I_i \in U_j$ and $a_{i,j} = 0$ otherwise.

De Werra [22] showed that any hypergraph with totally unimodular incidence matrix admits a balanced k-coloring. It is well-known [19] that a 0–1-matrix is totally unimodular if it has the consecutive-ones property, i. e., if there is a permutation of its columns such that all 1-entries appear consecutively in every row. The incidence matrix A of H has this property: If we order the U_j in increasing order of intersection points, then the entries $a_{i,j} = 1$ appear consecutively in each row. ◀

2.2 Algorithm for Two Colors

In this section, we present an algorithm that constructs a balanced 2-coloring in polynomial time. Since the algorithm produces a valid solution for every possible instance, Theorem 1 for $k = 2$ also follows from this algorithmic result. We note in passing that a polynomial-time algorithm can also be obtained by solving a simple Integer Linear Program (ILP) with a totally unimodular constraint matrix. However, this gives a much worse running time bound of $O(n^5 / \log n)$ [3].

The main idea of our algorithm is to simplify the structure of the instance such that the remaining intervals have start- and endpoints occurring pairwise. We then build a constraint graph that has the intervals as vertices. Finally, a proper 2-coloring of the constraint graph induces a solution to the problem.

The start- and endpoints of the intervals are called *events*. A *region* is an interval spanned by two consecutive events and is called even (odd) if it is contained in an even (odd) number of input intervals.

▶ **Theorem 2.** *For any set $\mathcal{I}$ of n intervals, there is a balanced 2-coloring that can be constructed in $O(n \log n)$ time.*

Proof. W. l. o. g. we can assume that the start- and endpoints of the input intervals are pairwise disjoint. If not, a new instance can be obtained by repeatedly increasing one of the coinciding start- or endpoints by $\epsilon/2$, where ϵ is the minimum size of a region. Since the new instance includes a corresponding region for every region of the original instance, a balanced coloring for the new instance is a balanced coloring for the old instance (the converse is not true).

Observe that a coloring that is balanced on all even regions is also balanced on all odd regions. This is because odd regions only differ by one interval from a neighboring even region. Thus, the task reduces to constructing a balanced coloring of the even regions. Since between two consecutive even regions, exactly two events occur, it suffices to consider only pairs of consecutive events enclosing odd regions.

If a pair of events consists of the start- and endpoint of the same interval, this interval is assigned any color and is removed from the instance. If a pair consists of start- and endpoint of different intervals, these intervals are removed from the instance and substituted by a new (minimal) interval that covers their union. In a final step of the algorithm, both intervals will be assigned the color of their substitution. The remaining instance consists solely of pairs of events where two intervals start or two intervals end. Clearly, a balanced coloring has to assign opposite colors to the corresponding two intervals of such a pair, and any such assignment yields a balanced coloring.

The remaining pairs of events induce a *constraint graph*. Every vertex corresponds to an interval, and an edge is added between two vertices if there is a pair of events containing both startpoints or both endpoints. Finding a proper vertex two-coloring of this graph gives a balanced 2-coloring. The constraint graph is bipartite: Each edge can be labeled by "$\vdash$" or "$\dashv$" if it corresponds to two start- or endpoints, respectively. Since each interval is incident to exactly two edges, any path must traverse $\vdash$- and $\dashv$-edges alternatingly. Therefore, every cycle must be of even length and hence the graph is bipartite. Thus, a proper vertex two-coloring of the constraint graph can be found in linear time by depth-first search.

Sorting events takes $O(n \log n)$ time. Creation of the constraint graph and coloring it takes linear time. ◀

Note that if intervals are given already sorted, or interval endpoints are described by small integers, then the above algorithm can even find a balanced 2-coloring in linear time.

2.3 Algorithms for k Colors

In this section, we extend the results of the previous section to an arbitrary number of colors k and show that a balanced interval k-coloring can be found in polynomial time.

A first polynomial time algorithm can be obtained using a construction by de Werra [22]: Start with an arbitrary coloring and find two colors i and j for which $\max_x |c_i(x) - c_j(x)|$ is maximal. Use the algorithm from Section 2.2 to find a balanced 2-coloring of all intervals that currently have color i or j and recolor them accordingly. Repeat until the coloring is balanced. This algorithm has running time $O(n \log n + k^2 n)$, because sorting intervals is needed only once for the above algorithm for 2 colors, and there are at most $\binom{k}{2}$ recolorings necessary.

In the following, we present an alternative algorithm for k colors, which is faster than $O(n \log n + k^2 n)$. We will first give an overview of the argument, and then a more formal description.

As in Section 2.2, we assume w. l. o. g. that all start- and endpoints are pairwise disjoint. The idea is to scan the events in order, beginning with the smallest, and to capture dependencies in k-tuples of intervals that indicate pairwise different colors. That is, we reduce the MINIMUM IMBALANCE INTERVAL k-COLORING instance to an instance of STRONG HYPERGRAPH COLORING, formally defined as follows.

STRONG HYPERGRAPH COLORING
Instance: A ground set X, a family $\mathcal{S}$ of constraints $S_1, \dots, S_n \subseteq X$, and an integer k.
Task: Find a k-coloring $\chi : X \to K$ with $\forall\, 1 \le i \le n : x, y \in S_i, x \neq y \Rightarrow \chi(x) \neq \chi(y)$.

For example, in the special case that each block of k consecutive events consists only of start- or only of endpoints, the constraints that the corresponding k intervals have to be differently colored will capture the whole solution. As we will see below, also different interval nesting structures can be captured by such constraints.

STRONG HYPERGRAPH COLORING is NP-hard in general [1]. However, each interval will occur in at most two constraints, corresponding to its start- and endpoint. Thus, we can further reduce the STRONG HYPERGRAPH COLORING instance to EDGE COLORING, where the goal is to color edges of a multigraph such that the edges incident to each vertex are all differently colored. More formally:

EDGE COLORING
Instance: A multigraph $G = (V, E)$ and an integer k.
Task: Find a k-coloring $\chi : E \to K$ with $\forall e, e' \in E, e \cap e' \neq \emptyset, e \neq e' \Rightarrow \chi(e) \neq \chi(e')$.

It is easy to see that any instance $(X, \mathcal{S}, k)$ of STRONG HYPERGRAPH COLORING where each element of X occurs in at most two constraints can be reduced to EDGE COLORING as $(G = (\mathcal{S}, E), k)$, where for each $x \in X$ that occurs in S_i and S_j with $i \neq j$, we add the edge $\{S_i, S_j\}$ to E. For our case, a constraint corresponds to a vertex, and an interval corresponds to an edge that connects the two constraints it occurs in. An edge coloring with k colors will thus provide a balanced interval k-coloring.

Clearly, an edge coloring of a multigraph with maximum degree Δ needs at least Δ colors. Finding an edge coloring of minimum size is NP-hard in general [14]. However, Kőnig [16] showed that for bipartite multigraphs, Δ colors in fact always suffice. Further, an edge coloring of a bipartite multigraph with m edges can be found in $O(m \log \Delta)$ time [8]. The multigraph we will construct has maximum degree k and is bipartite. Thus, a balanced interval k-coloring always exists and can be found in polynomial time.

We now describe the construction of the constraints and give a more rigorous description of the results. From a MINIMUM IMBALANCE INTERVAL k-COLORING instance $\mathcal{I}$, we construct a STRONG HYPERGRAPH COLORING instance $(\mathcal{I}, \mathcal{S}, k)$ over the ground set of the intervals. The algorithm scans the set of events in order, beginning with the smallest. It keeps a set of active events and adds constraints to $\mathcal{S}$. The set of active events will be cleared at each region where the number of intervals is 0 modulo k. Thus, the active events always describe the change from a situation where each color occurs the same number of times.

The construction of the constraints can be visualized with a decision tree, depicted for the example $k = 4$ in Figure 1. At the beginning, the set of active events is empty (which corresponds to the root of the decision tree). Whenever the set of events is empty, the algorithm branches into two cases depending on the type of the next event. Both branches are equivalent, with the roles of start- and endpoints interchanged. Therefore, assume the next event is the start of an interval (depicted as $\overset{a}{\vdash}$ on the left branch). It is added to the active set. This continues until either k startpoints of intervals $I_1, \dots, I_k$ are added, or an endpoint is encountered. In the first case, the constraint

$$(I_1, \dots, I_k) \tag{1}$$

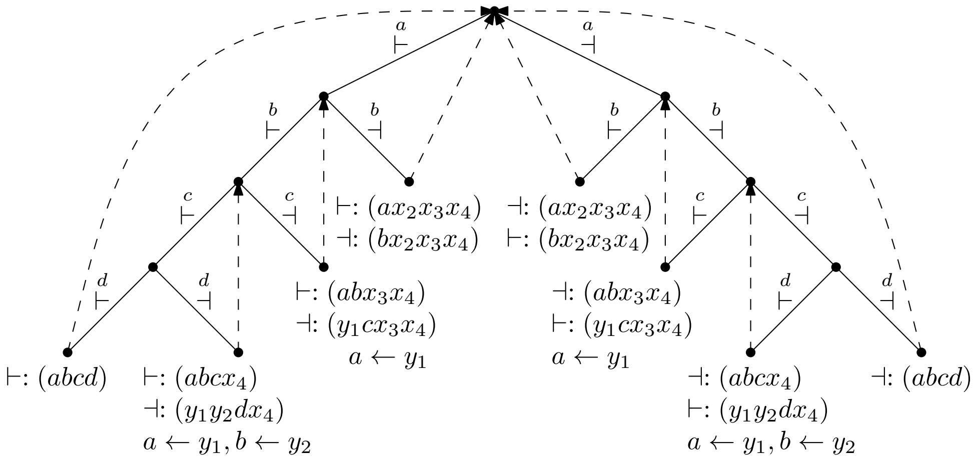


Figure 1 Tracking of active events for $k = 4$

is constructed and the set of active events is cleared (dashed arrow returning to the root). In the second case, assume the startpoints of intervals $I_1, \ldots, I_j$ have been added and the endpoint of interval I_{j+1} is encountered. Then, the two constraints

$$(I_1, \ldots, I_j, x_{j+1}, \ldots, x_k) \tag{2}$$

$$(y_1, \ldots, y_{j-1}, I_{j+1}, x_{j+1}, \ldots, x_k) \tag{3}$$

are constructed. Here, $x_{j+1}, \ldots, x_k$ and $y_1, \ldots, y_{j-1}$ are new "virtual" intervals that have not been used in previous constraints. That is, they do not correspond to actual intervals of the real line and only serve as placeholders in the active set. Furthermore, the startpoints of the intervals $I_1, \ldots, I_j$ are replaced by the startpoints of $y_1, \ldots, y_{j-1}$ in the active set of events (indicated by the dashed arrows pointing one level higher in the decision tree).

We now prove the correctness of the two chained reductions.

▶ **Lemma 3.** *A solution to the* Strong Hypergraph Coloring *instance* $(\mathcal{I}, \mathcal{S}, k)$*, constructed as described above, yields a balanced* k*-coloring for* $\mathcal{I}$*.*

Proof. Recall that a region is an interval spanned by two consecutive events. The proof is by induction over all regions, in the order of events. At each region, we count the number of times each of the k colors is used among the intervals containing the region. We will show that these counters differ by at most one, i. e., the coloring is balanced. Clearly, all counters are equal to zero before the first event.

In particular, we show that in regions where the number of intervals is 0 modulo k, all counters are equal, and that between these regions the counters change by at most one and all in the same direction. We distinguish the same cases as in the construction, limiting the discussion to the case that the event first added to the empty set of active events is a startpoint.

If k startpoints of intervals $I_1, \ldots, I_k$ are encountered, there is a constraint of the form (1), which ensures that all of them have different colors. Therefore, at each startpoint, a different counter increases by one. Hence, the counters differ by at most one in all regions up to the startpoint of I_k, and are all equal in the region beginning with the startpoint of I_k.

Consider that only $j < k$ startpoints of the intervals $I_1, \ldots, I_j$ are encountered. Since these startpoints were added to the set of active events during the construction, the intervals $I_1, \ldots, I_j$ do all occur in one constraint. This constraint forces them to have different colors, and therefore the colors of $I_1, \ldots, I_j$ are exactly the colors with increased count. Now, we distinguish two subcases

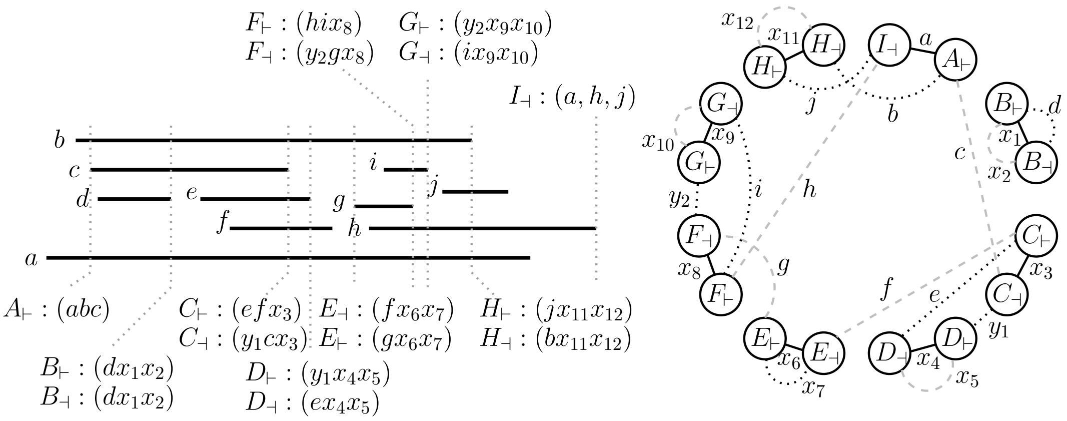


Figure 2 Complete example of the constraints and bipartite EDGE COLORING instance for $k = 3$ with a valid coloring.

depending on the next event. If the next event is a startpoint of some interval, denoted by I_{j+1}, it will also be part of the same constraint. Hence, I_{j+1} has a different color whose count is not yet increased. The second subcase is that the next event is the endpoint of some interval, denoted by I_{j+1}. The interval I_{j+1} is forced to have the same color as one of the intervals $I_1, \dots, I_j$. This is because there is a constraint of the form (2) that collects all other colors in variables $x_{j+1}, \dots, x_k$, which occur together with I_{j+1} in a constraint of the form (3). Thus, the previously increased counter for the color of I_{j+1} decreases again. Because of the constraint of the form (2), the virtual intervals $y_1, \dots, y_{j-1}$ must have all of the colors of $I_1, \dots, I_j$ except for the color of I_{j+1}. Hence, the colors of $y_1, \dots, y_{j-1}$ are exactly all remaining colors with increased count. Therefore, we are in the same situation as before encountering the endpoint of I_{j+1}. By repeating the above argument, the claim follows in this case.

If the event first added is an endpoint, the argument is symmetrical. ◀

▶ **Lemma 4.** *A* STRONG HYPERGRAPH COLORING *instance* $(\mathcal{I}, \mathcal{S}, k)$ *constructed as described above can be reduced to a bipartite* EDGE COLORING *instance.*

Proof. We need to show that each interval occurs in at most two constraints. Once this is proved, it is possible to build a multigraph with the constraints as vertices and edges between them if they share a common interval. Further, it has to be shown that this multigraph is bipartite. To show both parts at once, we color the constraints in $\mathcal{S}$ with the two colors $\vdash$ and $\dashv$. It then suffices to show that every interval can occur in at most one $\vdash$-constraint and in at most one $\dashv$-constraint.

Consider Figure 2 for an illustration of the constructed constraints and the respective bipartite EDGE COLORING instance.

We color a constraint with $\vdash$ if all involved events belonging to nonvirtual intervals are startpoints, and with $\dashv$ if all these events are endpoints (see Figure 1). All nonvirtual intervals therefore occur in exactly two constraints, constructed when the start- and endpoint get removed from the active set of events. A virtual x-interval always occurs in a pair of subsequent differently colored constraints, and is not used anywhere else. For the left branch of the decision tree, a virtual y-interval occurs first in a $\dashv$-constraint, and its startpoint is then added to the list of active events. Then, it will be used in a constraint of type either (1) or (2), both of which are of type $\vdash$. The argument is symmetrical for the right branch of the decision tree. ◀

▶ **Theorem 5.** *Every set of n intervals $\mathcal{I}$ has a balanced k-coloring for any $k \in \mathbb{N}$, and it can be found in $O(n \log n + kn \log k)$ time.*

Proof. By Lemmas 3 and 4, MINIMUM IMBALANCE INTERVAL k-COLORING can be reduced to EDGE COLORING with fixed k in a bipartite multigraph. The maximum degree of this multigraph is k, since by construction no constraint has more than k elements. By Kőnig's theorem [16] existence follows.

To be able to process the events, they have to be sorted in $O(n \log n)$ time. We have $O(kn)$ virtual intervals and thus $O(kn)$ edges in the EDGE COLORING instance. Finding an edge k-coloring for this multigraph with maximum degree k can be done in $O(kn \log k)$ time [8]. ◀

Note that the EDGE COLORING algorithm by Cole, Ost, and Schirra [8] uses quite involved data structures. In practice, it might be preferable to use the much simpler algorithm by Alon [2] running in $O(m \log m)$ time for an m-edge graph, which gives a worst-case bound of $O(kn \log n)$. An implementation of our algorithm in Python using a simple edge coloring algorithm based on augmenting paths can be found at http://www2.informatik.hu-berlin.de/~hueffner/intcol.py.

For an extension, recall that a matrix has the consecutive-ones property if there is a permutation of its columns such that all 1-entries appear consecutively in every row. Such a permutation can be found in linear time by the PQ-algorithm [7]. Given such a matrix, it is straightforward to construct an instance of MINIMUM IMBALANCE INTERVAL k-COLORING.

▶ **Theorem 6.** *For any hypergraph H with an $n \times m$ incidence matrix having the consecutive ones property, a balanced k-coloring can be found in $O(nm + kn \log k)$ time.*

2.4 Arcs of a Circle

In a periodic setting, the tasks $\mathcal{I}$ might be better described by a set of arcs of a circle rather than a set of intervals. In this case, there are instances that require an imbalance of two (e. g., three arcs that intersect exactly pairwise and $k = 2$). We show that two is also an upper bound and a coloring with maximal imbalance two can be found in polynomial time.

▶ **Theorem 7.** *The maximal imbalance for arcs of a circle is two, and finding a coloring with imbalance at most two can be done in $O(n \log n + kn \log k)$ time.*

Proof. Define a point on the circle, called zero, and consider counterclockwise orientation. We build an instance of MINIMUM IMBALANCE INTERVAL k-COLORING by "unfolding" the circle at zero in the following way. Consider only arcs that do not span the full circle. Map all such arcs not containing zero to intervals of same length at the same distance right of zero on the real line. Map the arcs containing zero to intervals of same length such that the positive part of the interval has the same length as the part of the arc in counterclockwise direction from zero. Finally, map the arcs containing the full circle to intervals spanning all of the instance constructed so far. Use the above algorithm to obtain a coloring of the intervals with imbalance at most one at every point. By reversing the mapping, the obtained coloring of the arcs has imbalance at most two (each point on the circle is mapped to at most two points of the real line). ◀

2.5 Online Algorithms

In load balancing problems, it is often more realistic to assume an online scenario, where not all information is known in advance, but is rather arriving piece-by-piece, and irrevocable decisions have to be made immediately. In our setting, this means that intervals arrive in order of their startpoint, including the information of their endpoint, and a color has to be assigned to them immediately.

The problem of finding a proper coloring (i. e., a coloring where no two intersecting intervals have the same color) of intervals in an online setting has found considerable interest [15, 11]. In these works, the objective is to use a minimum number of colors. In contrast, we consider a fixed number of colors and the minimization of the imbalance. We show that in contrast to the offline scenario, here the imbalance can become arbitrarily large.

▶ **Theorem 8.** *In online* MINIMUM IMBALANCE INTERVAL k-COLORING, *the imbalance is unbounded.*

Proof. We first consider the case $k = 2$ with colors "+1" and "−1". Denote the signed imbalance $\mathrm{simb}(x)$ to be the sum of the colors of the intervals containing x. Note that $\mathrm{imb}(x) = |\,\mathrm{simb}(x)|$.

In the following, we outline how a sequence of intervals can be constructed such that no online algorithm can yield a bounded imbalance. Initially, $\mathrm{simb} \equiv 0$. Set $L = [0, 1]$ and $R = [2, 3]$. Let L_ℓ, R_ℓ, L_r, and R_r denote the start- and endpoints of the current L and R, respectively. Repeat the following steps.

- Present the interval $[(L_\ell + L_r)/2, (R_\ell + R_r)/2]$ to the online algorithm.
- If it chooses color $+1$, set $R \leftarrow [R_\ell, (R_\ell + R_r)/2]$, else $R \leftarrow [(R_\ell + R_r)/2, R_r]$.
- Set $L \leftarrow [(L_\ell + L_r)/2, L_r]$.

This sequence of intervals is legal, since the startpoints increase strictly monotonously. In each repetition, if the algorithm chooses color $+1$, the signed imbalances in L and R increase by one. If the algorithm chooses -1, the signed imbalance decreases by one in L and remains unchanged in R, i. e., the difference of the signed imbalance in L and R increases. Therefore, the signed imbalance diverges in L or R. Since the imbalance is the absolute value of the signed imbalance, it becomes unbounded.

The construction easily generalizes to $k > 2$ colors. We only track two arbitrary colors, and whenever the algorithm assigns an untracked color to an interval, we present the same interval (with a slightly increased startpoint) again, forcing it to eventually assign a tracked color or to produce unbounded imbalance. ◀

3 Hardness of Generalizations

We consider several generalizations of MINIMUM IMBALANCE INTERVAL k-COLORING and show that they are NP-hard. Note that the hardness results of Biedl et al. [6] do not apply to the problems we consider here.

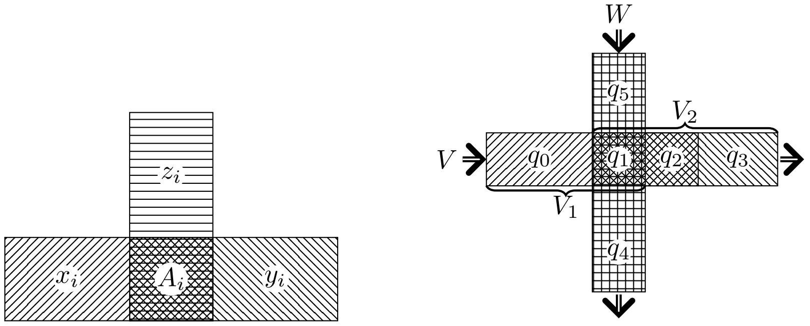
d-Dimensional Boxes. Gyárfás and Lehel [13] suggest to examine d-dimensional boxes as generalizations of intervals for coloring problems. The problem MINIMUM IMBALANCE d-BOX k-COLORING has as input an integer k and a set $\mathcal{I} = \{I_1, \ldots, I_n\}$ of n d-dimensional boxes $I_i = ([\ell_{i,1}, r_{i,1}], [\ell_{i,2}, r_{i,2}], \ldots, [\ell_{i,d}, r_{i,d}])$ for $1 \le i \le n$.

For every point $x = (x_1, \ldots, x_d)$, let $S(x)$ be the set of boxes that include x, i. e., $S(x)$ contains all the elements I_i such that $\ell_{i,j} \le x_j \le r_{i,j}$ for all $1 \le j \le d$. For a coloring $\chi : \mathcal{I} \to K$, a color i, and a point x, let $c_i(x)$ be the number of boxes in $S(x)$ of color i. With the analog definition of imbalance $\mathrm{imb}(\chi)$ and balance for d-dimensional boxes, the problem statement becomes:

MINIMUM IMBALANCE d-BOX k-COLORING
Instance: A set $\mathcal{I}$ of d-dimensional boxes.
Task: Find a k-coloring χ with minimal $\mathrm{imb}(\chi)$.

First note that, unlike for the case $d = 1$, a balanced coloring may not exist: already for three rectangles, some instances require imbalance two. Hence, we also have a related decision problem:

BALANCED d-BOX k-COLORING
Instance: A set $\mathcal{I}$ of d-dimensional boxes.
Question: Is there a balanced k-coloring χ?



(a) Gadget for clause $C_i = (x_i, y_i, z_i)$ (b) Gadget for crossing chains V and W

Figure 3 The gadgets of the reduction.

We show that for all $d \geq 2$ and $k \geq 2$, it is NP-complete to decide BALANCED d-BOX k-COLORING. This clearly implies NP-hardness of MINIMUM IMBALANCE d-BOX k-COLORING.

▶ **Theorem 9.** BALANCED d-BOX k-COLORING *is NP-complete for any* $d \geq 2$ *and any* $k \geq 2$.

We will reduce from NOT-ALL-EQUAL 3SAT (NAE-3SAT) [18]. Note that the classic definition of NAE-3SAT [12] allows negated variables. However, this is not needed to make the problem NP-complete [18]. Thus, in the sequel, we will assume that all variables occur only non-negatedly.

> NOT-ALL-EQUAL 3SAT (NAE-3SAT)
> **Instance:** A Boolean formula with clauses $C_1, \ldots, C_m$, each having at most 3 variables.
> **Question:** Is there a truth assignment such that in every clause, not all variables have the same value?

We first consider $k = 2$ and then generalize to arbitrary k. We present the gadgets of the reduction, then show how they are combined together, and conclude by proving correctness.

For each clause $C_i = (x_i, y_i, z_i)$, we construct a *clause gadget* comprised of three rectangles (see Figure 3a). Note that all three rectangles overlap in region A_i, and only there. Then we also construct a separate rectangle r_j for every variable. Finally, we connect each r_j to all rectangles that appear in a clause gadget, and correspond to the same variable as r_j. We do this by a chain with odd number of rectangles. This ensures that in any balanced 2-coloring, r_j and the corresponding rectangle in the clause gadget have the same color. If two chains need to cross, we introduce a *crossing gadget* as seen in Figure 3b. Three rectangles are relevant for the crossing of two chains V and W. The first is V_1 and contains areas q_0, q_1, and q_2, the second is V_2, containing q_1, q_2, and q_3. Both V_1 and V_2 belong to chain V. The last rectangle contains areas q_1, q_4 and q_5 and belongs to chain W. Note that the crossing does not induce any dependencies on the colorings between chains V and W. See Figure 4 for a construction of an instance for BALANCED 2-BOX 2-COLORING.

Observe that the above construction only requires a number of rectangles polynomial in the size of the NAE-3SAT instance.

▶ **Lemma 10.** BALANCED d-BOX 2-COLORING *is NP-complete for any* $d \geq 2$.

Proof. The problem is in NP, since feasibility of a color assignment can be checked in polynomial time. For NP-hardness, we show that a NAE-3SAT instance is satisfiable if and only if the answer to the corresponding BALANCED 2-BOX 2-COLORING instance is "yes". This also implies NP-completeness for every $d \geq 2$ by taking intervals of length 0 in higher dimensions.

($\Rightarrow$) Assume that there is a satisfying assignment of the NAE-3SAT instance. Then, color the rectangles r_j according to the truth values of their corresponding variables. This coloring can be

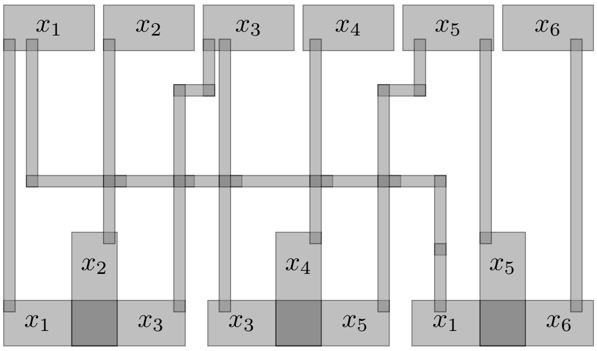


Figure 4 Example for NAE-3SAT instance $(x_1, x_2, x_3), (x_3, x_4, x_5), (x_1, x_5, x_6)$.

easily extended to all the rectangles by alternatively coloring rectangles along a chain (and crossings) starting from each r_j and ending at a clause gadget. It remains to show that $\text{imb}(x) \leq 1$ holds for every point $x \in A_i$ for all $1 \leq i \leq m$. Consider A_i corresponding to clause $C_i = (x_i, y_i, z_i)$. The three rectangles that intersect at A_i have the colors corresponding to the truth values of their variables x_i, y_i, and z_i in the solution of NAE-3SAT. Since the three variables do not have all the same truth value, the three rectangles cannot have all the same color, and $\text{imb}(x) \leq 1$.

($\Leftarrow$) Assume that we have a balanced 2-coloring for the constructed BALANCED 2-BOX 2-COLORING instance. Consider only the clause gadgets. We have already observed that rectangles that correspond to the same variable and appear in clause gadgets must have the same color. We can assign the truth values of the variables according to the colors in the corresponding rectangles. Since in no A_i all three rectangles have the same color, in no C_i all three variables have the same truth value, yielding a feasible solution for NAE-3SAT. ◀

Proof of Theorem 9. First apply the construction for BALANCED 2-BOX 2-COLORING and call its rectangles *reduction rectangles*. Then add $k - 2$ additional rectangles that fully contain the construction and all intersect at least in one point outside the construction; these are called *cover rectangles*. By the latter property, cover rectangles must have distinct colors in any balanced coloring. Observe that each reduction rectangle contains some point that does not intersect with other reduction rectangles but only with all the cover rectangles. This implies that the reduction rectangles have available only the two colors not used by the cover rectangles. We conclude that the problem of k-coloring the constructed instance is equivalent to the problem of 2-coloring only the reduction rectangles. ◀

Further Generalizations. The weighted version, where intervals have weights and the weighted imbalance is to be minimized, is NP-complete by reduction from PARTITION. Furthermore, the variant with multiple intervals [13] is NP-complete by reduction from NAE-3SAT. Both hardness results generalize to higher dimensions. Proofs are omitted due to space constraints.

4 Open Questions

- We have given a polynomial time algorithm for k-coloring hypergraphs with the consecutive-ones property, i. e., a special case of a totally unimodular incidence matrix. It would be interesting to generalize to arbitrary totally unimodular incidence matrices.
- For arcs of a circle, we have shown how to find a coloring with imbalance at most two in polynomial time, but it is not clear how to find an optimal one.

- It remains open how large the imbalance can become for d-dimensional boxes, and whether we can find polynomial-time approximations for it. We were not able to find an instance requiring an imbalance greater than 2 for the 2-dimensional case.

References

1 G. Agnarsson and M. M. Halldórsson. Strong colorings of hypergraphs. In *Proc. 2nd WAOA*, volume 3351 of *LNCS*, pages 253–266. Springer, 2005.

2 N. Alon. A simple algorithm for edge-coloring bipartite multigraphs. *Information Processing Letters*, 85(6):301–302, 2003.

3 K. M. Anstreicher. Linear programming in $O(\frac{n^3}{\ln n}L)$ operations. *SIAM Journal on Optimization*, 9(4):803–812, 1999.

4 N. Bansal. Constructive algorithms for discrepancy minimization. In *Proc. 51st FOCS*. IEEE Computer Society, 2010. To appear. Also in arXiv:1002.2259v4.

5 J. Beck and T. Fiala. "Integer making" theorems. *Discrete Applied Mathematics*, 3(1):1–8, 1981.

6 T. C. Biedl, E. Čenek, T. M. Chan, E. D. Demaine, M. L. Demaine, R. Fleischer, and M.-W. Wang. Balanced k-colorings. *Discrete Mathematics*, 254(1–3):19–32, 2002.

7 K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13(3):335–379, 1976.

8 R. Cole, K. Ost, and S. Schirra. Edge-coloring bipartite multigraphs in $O(E \log D)$ time. *Combinatorica*, 21(1):5–12, 2001.

9 B. Doerr. *Integral Approximation*. Habilitationsschrift, Christian-Albrechts-Universität zu Kiel, 2005.

10 B. Doerr and A. Srivastav. Multicolour discrepancies. *Combinatorics, Probability and Computing*, 12:365–399, 2003.

11 L. Epstein. Online interval coloring. In M.-Y. Kao, editor, *Encyclopedia of Algorithms*, pages 594–598. Springer, 2008.

12 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

13 A. Gyárfás and J. Lehel. Covering and coloring problems for relatives of intervals. *Discrete Mathematics*, 55(2):167–180, 1985.

14 I. Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981.

15 H. A. Kierstead and W. T. Trotter. An extremal problem in recursive combinatorics. *Congressus Numerantium*, 33:143–153, 1981.

16 D. Kőnig. Gráfok és alkalmazásuk a determinánsok és a halmazok elméletére [in Hungarian: Graphs and their application to determinant theory and set theory]. *Matematikai és Természettudományi Értesítő*, 34:104–119, 1916.

17 J. Matoušek. *Geometric Discrepancy: An Illustrated Guide*, volume 18 of *Algorithms and Combinatorics*. Springer, 1999.

18 T. J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th STOC*, pages 216–226. ACM, 1978.

19 A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, 1986.

20 J. Spencer. Six standard deviations suffice. *Transactions of the American Mathematical Society*, 289(2):679–706, 1985.

21 A. Srinivasan. Improving the discrepancy bound for sparse matrices: Better approximations for sparse lattice approximation problems. In *Proc. 8th SODA*, pages 692–701. ACM-SIAM, 1997.

22 D. de Werra. Equitable colorations of graphs. *Revue Française d'Informatique et de Recherche opérationnelle*, R-3:3–8, 1971.

Symmetric Determinantal Representation of Weakly-Skew Circuits

Bruno Grenet[1,2], Erich L. Kaltofen[*3], Pascal Koiran[1,2], and Natacha Portier[†1,2]

1 LIP, UMR 5668, ENS de Lyon – CNRS – UCBL – INRIA
École Normale Supérieure de Lyon, Université de Lyon
[Bruno.Grenet,Pascal.Koiran,Natacha.Portier]@ens-lyon.fr
2 Department of Computer Science, University of Toronto
3 Dept. of Mathematics, North Carolina State University,
Raleigh, North Carolina 27695-8205, USA
kaltofen@math.ncsu.edu
http://www.kaltofen.us

Abstract

We deploy algebraic complexity theoretic techniques for constructing symmetric determinantal representations of weakly-skew circuits, which include formulas. Our representations produce matrices of much smaller dimensions than those given in the convex geometry literature when applied to polynomials having a concise representation (as a sum of monomials, or more generally as an arithmetic formula or a weakly-skew circuit). These representations are valid in any field of characteristic different from 2. In characteristic 2 we are led to an almost complete solution to a question of Bürgisser on the VNP-completeness of the partial permanent. In particular, we show that the partial permanent cannot be VNP-complete in a finite field of characteristic 2 unless the polynomial hierarchy collapses.

1998 ACM Subject Classification F.1.1, F.2.1, I.1.1, I.1.2

Keywords and phrases algebraic complexity, determinant and permanent of symmetric matrices, formulas, skew circuits, Valiant's classes

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.543

1 Introduction

1.1 Motivation

A linear matrix expression (symmetric linear matrix form, affine symmetric matrix pencil) is a symmetric matrix with the entries being linear forms in the variables $x_1, \dots, x_n$ and real number coefficients:

$$A(x_1, \dots, x_n) = A_0 + x_1 A_1 + \cdots + x_n A_n, \quad A_i \text{ symmetric in } \mathbb{R}^{t \times t}. \tag{1}$$

A linear matrix inequality (LMI) restricts to those values $\xi_i \in \mathbb{R}$ of the x_i such that $A(\xi_1, \dots, \xi_n) \succeq 0$, i.e., is positive semidefinite. The set of all such values defines a spectrahedron.

[*] This material is based on work supported in part by the National Science Foundation under Grants CCF-0830347 and CCF-0514585.

[†] This material is based on work supported in part by the European Community under contract PIOF-GA-2009-236197 of the 7th PCRD.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 543–554
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

A *real zero polynomial* is a polynomial p with real coefficients such that for every $x \in \mathbb{R}^n$ and every $\mu \in \mathbb{C}$, $p(\mu x) = 0$ implies $\mu \in \mathbb{R}$. The Lax conjecture and generalized Lax conjecture seek for representations of real zero polynomials $f(x_1, \ldots, x_n)$ (1) with $f = \det(A)$ and $A_0 \succeq 0$. This is in fact an equivalent formulation of the original Lax conjecture which was stated in terms of hyperbolic polynomials (see [11] for this equivalence). Furthermore, the matrices are required to have dimension d where d is the degree of the polynomial. For $n = 2$ such representations always exist while a counting argument shows that this is impossible for $n > 2$ [8] (actually, the authors of [11] give the first proof of the Lax conjecture in its original form based on the results of [8]). Two generalizations have been suggested to avoid this counting argument: first, it was suggested to remove the dimension constraint and allow for bigger matrices, and second, to permit representations of some power of the input polynomial. Counterexamples to both generalizations have recently been constructed [3].

Another relaxation is to drop the condition $A_0 \succeq 0$ and represent any f as $\det(A)$ [7, 16]. However, the purely algebraic construction of [16] leads to exponential matrix dimensions t. Here we continue the line of work initiated by [7, 16] but we proceed differently by symmetrizing the complexity theoretic construction by Valiant [18]. Our construction yields smaller dimensional matrices not only for polynomials represented as sums of monomials but also for polynomials represented by formulas and weakly-skew circuits [14, 9]. Even though in the most general case the bounds we obtained are slightly worse than Quarez's [16], in a lot of interesting cases such as polynomials with a polynomial size formula or weakly-skew circuit, or in the case of the permanent, our constructions yield much smaller matrices [5, Section 4].

Our constructions are valid for any field of characteristic different from 2. For fields of characteristic 2, it can be shown that some polynomials (such as e.g. the polynomial $xy + z$) cannot be represented as determinants of symmetric matrices [6]. Note as a result that the 2-dimensional permanent $xw + yz$ cannot be "symmetrized" over characteristic 2 with any dimension. It would be interesting to exactly characterize which polynomials admit such a representation in characteristic 2. For the polynomial $x + y$, we have

$$x + y = \det(\begin{bmatrix} 0 & x & 0 & y & -1 \\ x & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ y & 0 & -1 & 0 & 1/2 \\ -1 & 0 & 0 & 1/2 & 0 \end{bmatrix}) = \det(\begin{bmatrix} x & 0 & 0 & 1 \\ 0 & y & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}),$$

where the first matrix is derived from our construction, but the second is valid over any commutative ring. It is easily shown that for every polynomial p, p^2 admits a symmetric determinantal representation in characteristic 2. This is related to a question of Bürgisser [4]: Is the partial permanent VNP-complete over fields of characteristic 2? We give an almost complete negative answer to this question.

Our results give as a by-product an interesting result, which was not known to the authors' knowledge: Let A be an $(n \times n)$ matrix with indeterminate coefficients (ranging over a field of characteristic different from 2); then there exists a symmetric matrix B of size $0(n^5)$ whose entries are the indeterminates from A and constants from the field such that $\det A = \det B$. This relies on the existence of a size-$O(n^5)$ weakly-skew circuit to compute the determinant of an $(n \times n)$ matrix [2, 14]. The size of B can be reduced to $O(n^4)$ if we replace the weakly-skew circuits from [2, 14] by the skew circuits of size $O(n^4)$ constructed by Mahajan and Vinay [13]. These authors construct an arithmetic branching program for the determinant with $O(n^4)$ edges,[1] and the arithmetic branching program can be evaluated by a skew circuit of size $O(n^4)$. After learning of our result, Meena Mahajan and Prajakta Nimbhorkar have noticed that the arithmetic branching program for the determinant can be transformed directly into a symmetric determinant of size $O(n^3)$ with techniques similar to the ones used in this paper. A detailed proof will appear in the full version of this paper.

Acknowledgments: We learned of the symmetric representation problem from Markus Schweighofer's ISSAC 2009 Tutorial
`http://www.math.uni-konstanz.de/~schweigh/presentations/dcssblmi.pdf`.

We thank Meena Mahajan for pointing out [13] and sketching the construction of a symmetric determinant of size $O(n^3)$ from a determinant of size n.

1.2 Known results and definitions

In his seminal paper Valiant [18] expressed the polynomial computed by an arithmetic formula as the determinant of a matrix whose entries are constants or variables. If we define the *skinny size* e of the formula as its number of arithmetic operations then the size of the matrix is at most $e + 2$. The proof uses a weighted digraph construction where the formula is encoded into paths from a source vertex to a target, sometimes known as an Algebraic or Arithmetic Branching Program [15, 1]. This theorem shows that every polynomial with a sub-exponential size formula can be expressed as a determinant with sub-exponential size formula, enhancing the prominence of linear algebra. A slight variation of the theorem is also used to prove the universality of the permanent for formulas which is one of the steps in the proof of its VNP-completeness. In a tutorial, von zur Gathen [21] gives another way to express a formula as a determinant: his proof does not use digraphs and his bound is $2e + 2$. Refining von zur Gathen's techniques, Liu and Regan [12] gave a construction leading to a $e + 1$ bound and an extra property: multiplications by constant are not counted in e.

In [17, 14], results of the same flavor were proved for a more general class of circuits, namely the *weakly-skew* circuits. Malod and Portier [14] can deduce from those results a fairly simple proof of the VQP-completeness of the determinant (under qp-projection). Moreover, they define a new class VP_{ws} of polynomials represented by polynomial-size weakly-skew circuits (with no explicit restriction on the degree of the polynomials) for which the determinant is complete under p-projection. (See [4, 14] for the definitions.) A formula is a circuit in which every vertex has out-degree 1 (but the output). This means in particular that the underlying digraph is a tree. A weakly-skew circuit is a kind of generalization of a formula, with a less constrained structure on the underlying digraph. For an arithmetic circuit, the only restriction on the digraph is the absence of directed cycles (that is the underlying digraph is a directed acyclic graph). A circuit is said weakly-skew if every multiplication gate α has the following property: the sub-circuit associated with one of its arguments β is connected to the rest of the circuit only by the arrow going from β to α. This means that the underlying digraph is disconnected as soon as the multiplication gate α is removed. In a sense, one of the arguments of the multiplication gate was separately computed for this gate.

Toda [17] proved that the polynomial computed by a weakly-skew circuit of skinny size e can be represented by the determinant of a matrix of size $(2e + 2)$. This result was improved by Malod and Portier [14]: The construction leads to a matrix of size $(m + 1)$ where m is the *fat size* of the circuit (*i.e.* its total number of gates, including the input nodes). Note that for a circuit in general and for a weakly-skew circuit in particular $m \leq 2e + 1$. The

[1] This bound can be found on p.11 of their paper.

latter construction uses negated variables in the matrix. It is actually possible to get rid of them [9]. Although the skinny size is well suited for the formulas, the fat size appears more appropriate for weakly-skew circuits. In Section 2, we symmetrize this construction so that a polynomial expressed by a weakly-skew circuit equals the determinant of a symmetric matrix. Our construction yields a size-$(2m+1)$ symmetric matrix.

Let us now give some formal definitions of the arithmetic circuits and related notions.

▶ **Definition 1.** An *arithmetic circuit* is a directed acyclic graph with vertices of in-degree 0 or 2 and exactly one vertex of out-degree 0. Vertices of in-degree 0 are called *inputs* and labelled by a constant or a variable. The other vertices, of in-degree 2, are labeled by $\times$ or $+$ and called *computation gates*. The vertex of out-degree 0 is called the *output*. The vertices of a circuit are commonly called *arithmetic gates* and its arcs *arrows*.

A (division-free) arithmetic circuit with constants in a field k and input variables $x_1, \ldots, x_n$ naturally computes a polynomial $f \in k[x_1, \ldots, x_n]$.

If α is a gate of a circuit C, the *sub-circuit associated to* α is the subgraph of C made of all the gates β such that there exists a oriented path from β to α in C, including α. The gates β and γ are called the *arguments* of α.

An arithmetic circuit is said *weakly-skew* if for any multiplication gate α, the sub-circuit associated to one of its arguments β is only connected to the rest of the circuit by the arrow going from β to α: it is called the *closed* sub-circuit of α. A gate which does not belong to a closed sub-circuit of C is said to be *reusable* in C. The reusability of a gate depends, of course, on the considered circuit C.

In our constructions, we shall use *graphs* and *digraphs*. In order to avoid any confusion between directed and undirected graphs, we shall exclusively use the term graph for undirected ones, and otherwise use the term digraph. It is well-known that cycle covers in digraphs are in one-to-one correspondence with permutations of the vertices and therefore that the permanent of the adjacency matrix of a digraph can be defined in terms of cycle covers of the graph. Let us now give some definitions for those facts, and see how it can be extended to graphs.

▶ **Definition 2.** A *cycle cover* of a digraph $G = (V, A)$ is a set of cycles such that each vertex appears in exactly one cycle. The *weight* of a cycle cover is defined to be the product of the weights of the arcs used in the cover. Let the *sign* of a vertex cover be the sign of the corresponding permutation of the vertices, that is $(-1)^N$ where N is the number of even cycles. Finally, let the *signed weight* of a cycle cover be the product of its weight and sign.

For a graph $G = (V, E)$, let $G^d = (V, A)$ be the corresponding symmetric digraph. Then a cycle cover of G is a cycle cover of G^d, and the definitions of weight and sign are extended to this case. In particular, if there is a cycle cover of G with a cycle $C = (u_1, \ldots, u_k)$, then a new cycle cover is defined if C is replaced by the cycle $(u_k, \ldots, u_1)$. Those two cycle covers are considered as different cycle covers of G.

▶ **Definition 3.** Let G be a digraph. Its *adjacency matrix* is the $(n \times n)$ matrix A such that $A_{i,j}$ is equal to the weight of the arc from i to j ($A_{i,j} = 0$ is there is no such arc). The definition is extended to the case of graphs, seen as symmetric digraphs. In particular, the adjacency matrix of a graph is symmetric.

▶ **Lemma 4.** *Let G be a (di)graph, and A its adjacency matrix. Then the permanent of A equals the sum of the weights of all the cycle covers of G, and the determinant of A is equal to the sum of the signed weights of all the cycle covers of G.*

Proof. The cycle covers are obviously in one-to-one correspondence with the permutations of the set of vertices, and the sign of a cycle cover is defined to match the sign of the corresponding permutation. Suppose that the vertices of V are $\{1,\ldots,n\}$ and let $A_{i,j}$ be the weight of the arc (i,j) in G. Let C a cycle cover and σ the corresponding permutation. Then it is clear that the weight of C is $A_{1,\sigma(1)}\cdots A_{n,\sigma(n)}$, hence the result. ◀

The validity of this proof for graphs follows from the definition of the cycle covers of a graph in terms of the cycle covers of the corresponding symmetric digraph. In the following, the notion of perfect matching is used. A *perfect matching* in a graph G is a set M of edges of G such that every vertex is incident to exactly one edge of M. The weight of a perfect matching is defined in this paper as the weight of the corresponding cycle cover (with length-2 cycles). This means that it is the product of the weights of the arcs it uses, or equivalently it is the square of the product of the weights of the edges it uses. Note that this is the square of the usual definition.

A *path* P in a digraph is a subset of vertices $\{u_1,\ldots,u_k\}$ such that for $1 \le i \le k-1$, there exists an arc from u_i to u_{i+1} with nonzero weight. The size $|P|$ of such a path is k.

2 Weakly-skew circuits

In this section, we extend the construction of [14] to the case of symmetric matrices: given a weakly-skew circuit computing a polynomial p, a symmetric matrix M which entries are variables and constants is built such that $p = \det M$. Malod and Portier [14] express a polynomial as a determinant of a non-symmetric matrix. Their construction relies on the construction of a digraph whereas ours relies on the construction of a (non-directed) graph. Recall that a weakly-skew circuit has several *reusable* gates. This means that when a weakly-skew circuit is recursively turned into a (di)graph, some vertices have to be *reusable*. This is ensured in [14] by the property that the digraph is acyclic. As we are dealing with a graph instead of a digraph, this cannot be used anymore. A solution to this problem is given in Lemma 6 by introducing the notion of acceptable paths: A path P in a graph G is said *acceptable* if $G \setminus P$ admits a cycle cover.

As in [14], the size bounds of the constructed matrix and graph are given in terms of the fat size of the weakly-skew circuit: the *fat size* of a circuit is its total number of gates, including the input gates. Note that one can refine these bounds using the notion of *green size* defined in the long version of this paper [5, Section 3.2]. Furthermore, if the polynomial is given as a formula instead of a weakly-skew circuit, it is possible to get tighter bounds [5, Section 2].

Let us fix a field k of characteristic different from 2 and a countable set $\bar{x} = \{x_1, x_2, \ldots\}$ of variables. The circuits we consider are supposed to have inputs in $k \cup \bar{x}$.

▶ **Theorem 5.** *Let f be a polynomial computable by a weakly-skew circuit of fat size m. Then there exists a symmetric matrix A of size at most $2m+1$ whose entries are inputs of the circuit and elements from $\{0,1,-1,1/2\}$ such that $f = \det A$.*

The proof relies on the following lemma. It applies to so-called *multiple-output* weakly-skew circuits. This generalization just consists in circuits for which there exist several out-degree-0 gates.

▶ **Lemma 6.** *Let C be a multiple-output weakly-skew circuit of fat size m. There exists a graph G with at most $2m+1$ vertices and a distinguished vertex s such that $|G|$ is odd, every cycle in G is even, and for every reusable gate $\alpha \in C$ there exists a vertex $t_\alpha \in G$ such that*

1. *Every s-t_α-path (whether acceptable or not) has an odd number of vertices;*
2. *For every acceptable s-t_α-path P in G, the subgraph $G \setminus P$ is either empty or has a unique cycle cover, which is a perfect matching of weight 1;*
3. *The following equality holds in G:*

$$\sum_{\substack{\text{acceptable} \\ s\text{-}t_\alpha\text{-path } P}} (-1)^{\frac{|P|-1}{2}} w(P) = f_\alpha \tag{2}$$

where f_α is the polynomial computed by the gate α.

Furthermore, the graph $G \setminus \{s\}$ has a unique cycle cover which is a perfect matching of weight 1.

Proof sketch. The graph G is built by induction on the (fat) size of the circuit. We only sketch here its construction. For a proof that G satisfies the conditions of the lemma, refer to [5, Lemma 4]. If α is a reusable gate of C, then t_α is said to be a reusable vertex of G.

A size-1 circuit is an input gate α with label x. The corresponding graph G has three vertices: s, t_α and an additional vertex v_α. There is an edge between s and v_α of weight x, and an edge between v_α and t_α of weight -1.

Let $m > 1$ and suppose that the lemma holds for any multiple-output weakly-skew circuit of size less than m. Let C be a multiple output weakly-skew circuit of size m, and α be any of its outputs.

If α is an input gate with label x, let $C' = C \setminus \{\alpha\}$ and G' the corresponding graph with a distinguished vertex s. The graph G is obtained from G' by adding two new vertices v_α and t_α, an edge of weight x between s and v_α and an edge of weight -1 between v_α and t_α (see Fig. 1). The vertex s is the distinguished vertex of G.

If α is an addition gate, let $C' = C \setminus \{\alpha\}$ and suppose that α receives arrows from gates β and γ. Note that β and γ are reusable. Let G' be the graph corresponding to C', and s be its distinguished vertex. G' contains two reusable vertices t_β and t_γ. The graph G is obtained by adding two vertices v_α and t_α, and the following edges: $t_\beta v_\alpha$ and $t_\gamma v_\alpha$ of weight 1, and $v_\alpha t_\alpha$ of weight -1 (see Fig. 3). If $\beta = \gamma$, then G' contains a vertex t_β, and we merge the two edges adjacent to t_β and t_γ into an edge $t_\beta v_\alpha$ of weight 2.

If α is a multiplication gate, α receives arrows from two distinct gates β and γ. Exactly one of those gates, say β, is not reusable and removing the gate α yields two disjoint circuits C_1 and C_2 (say β belongs to C_1 and γ to C_2). Let G_1 and G_2 be the respective graphs obtained by induction from C_1 and C_2, with distinguished vertices s_1 and s_2 respectively. The graph G is obtained as in Fig. 2 as the union of G_1 and G_2 where t_γ and s_1 are merged, the distinguished vertex s of G being the distinguished vertex s_2 of G_2, and t_α being equal to t_β. ◀

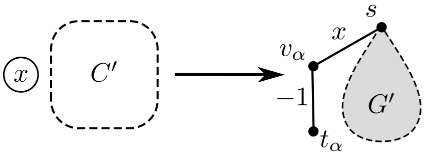


Figure 1 Input gate

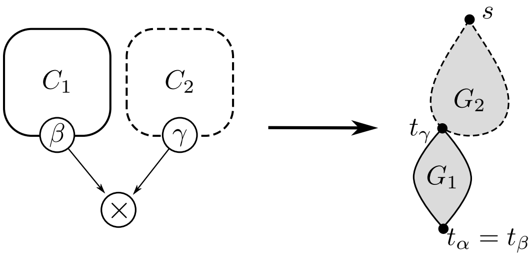


Figure 2 Multiplication gate

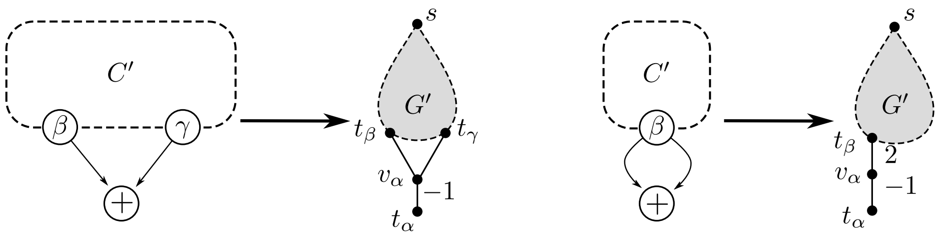


Figure 3 Both cases for an addition gate

Proof of Theorem 5. Let C be a weakly-skew circuit computing the polynomial f, and G be the graph built from C in Lemma 6. The circuit C has a unique output, and there exists in G a vertex t corresponding to this output. Let G' be the graph obtained from G by adding an edge between t and s of weight $\frac{1}{2}(-1)^{\frac{|G|-1}{2}}$.

There is no cycle cover of G' containing the 2-cycle st. Indeed, $|G' \setminus \{s,t\}|$ is odd and G contains only even cycles. This means that a cycle cover of G' contains a cycle made of a s-t-path plus (t,s) or a t-s-path plus (s,t). Let P be such a path. Then $G' \setminus P = G \setminus P$. Hence, by Lemma 6, there is exactly one cycle cover of $G' \setminus P$ and it is a perfect matching of weight 1. This means that there is a one-to-one correspondence between the cycle covers of G' and the paths from s to t or from t to s. There is also a one-to-one correspondence between the paths from s to t and the paths from t to s.

Let us recall that the sign of a cycle cover is the sign of the underlying permutation and its signed weight is the product of its sign and weight. Let C be a cycle cover of G' involving the s-t-path P. The previous paragraph shows that the weight of C equals $\frac{1}{2}(-1)^{\frac{|G|-1}{2}}w(P)$. As C has an odd cycle and a perfect matching, its sign is $(-1)^{|G\setminus P|/2}$, that is the number of couples in the perfect matching. The inverse cycle cover $\bar{C}$ of G' has the same signed weight as C. Hence the sum of the signed weights of all cycle covers of G' equals twice the sum over all s-t-paths P of $\frac{1}{2}(-1)^{\frac{|G|-1}{2}}(-1)^{\frac{|G\setminus P|}{2}}w(P) = \frac{1}{2}(-1)^{\frac{|P|-1}{2}}w(P)$. By Lemma 6, this equals f and Lemma 4 concludes the proof. ◀

3 Characteristic 2

In characteristic 2, the construction of Section 2 fails because of the scalar 1/2 it uses. Nevertheless, for a polynomial computable by a weakly-skew circuit, it is possible to represent, by the usual symmetrization, its square as the determinant of a symmetric matrix. On the other hand, as pointed out in the introduction representing the polynomial itself is not always possible. Related to these problems, the VNP-completeness of the partial permanent is also studied. Actually, we give an almost complete answer to an open question of Bürgisser [4, Problem 3.1] showing that if the partial permanent is complete in finite fields of characteristic 2, then the (boolean) polynomial hierarchy collapses. For any field of characteristic 2 (finite or infinite), we show that the VNP-completeness of this family would imply that every VNP family of polynomials has its square in VP_{ws}. This also seems unlikely to happen unless $\mathsf{VP}_{ws} = \mathsf{VNP}$.

Let G be an edge-weighted graph with vertices $\{v_1, \ldots, v_n\}$. Recall that the adjacency matrix A of G is the $(n \times n)$ symmetric matrix defined by $A_{ij} = A_{ji} = w_{ij}$ where w_{ij} is the weight of the edge v_iv_j. Suppose now that G is bipartite with two independent sets of vertices V_r and V_c of cardinality m and n respectively. Let $V_r = \{r_1, \ldots, r_m\}$ and $V_c = \{c_1, \ldots, c_n\}$.

The *biadjacency matrix* of G (also known as the *bipartite adjacency matrix*) is the $(m \times n)$ matrix B such that B_{ij} is the weight of the edge between r_i and c_j. This means that the rows of B are indexed by V_r and its columns by V_c. For a bipartite graph G of adjacency and biadjacency matrices A and B respectively,

$$A = \begin{pmatrix} 0 & B \\ B^t & 0 \end{pmatrix}.$$

Throughout this section, we shall use the usual definition of the weight of a partial matching: it is the product of the weights of the edges it uses.

3.1 Symmetric determinantal representation of the square of a polynomial

▶ **Lemma 7.** *Let G be an edge-weighted graph and A its adjacency matrix. In characteristic* 2*, the determinant of A is the sum of the weights of the cycle covers with cycles of length at most* 2*.*

Proof. Let us consider G as a symmetric digraph (that is an edge uv is seen as both arcs (u, v) and (v, u)). In Lemma 4, the signs of the cycle covers are considered. In characteristic 2, this is irrelevant. Therefore, the determinant of A is the sum of the weights of the cycle covers of G.

Let C be a cycle cover of G containing a (directed) cycle of length at least 3 denoted by $(v_1, v_2, \dots, v_k, v_1)$. One can change the direction of this cycle (as G is symmetric) and obtain a new cycle cover C' containing the same cycles as C, but $(v_k, v_{k-1}, \dots, v_1, v_k)$ instead of $(v_1, v_2, \dots, v_k, v_1)$. Clearly, the weights of C and C' are the same as the graph is symmetric. Therefore, when the determinant of A is computed in characteristic 2, the contributions of those two cycle covers to the sum cancel out. This shows that the determinant of a matrix in characteristic two is obtained as the sum of the weights of cycle covers with cycles of length 1 (loops) or 2. ◀

▶ **Proposition 8.** *Let p be a polynomial over a field of characteristic* 2*, represented by a weakly-skew circuit of fat size m. Then there exists a symmetric matrix A of size* $(2m + 2)$ *such that $p^2 = \det(A)$.*

Proof. Let C be a weakly-skew circuit representing a polynomial p over a field of characteristic 2. Let M be the matrix obtained by Malod and Portier's construction [14] such that $p = \det(M)$. Let G be the digraph represented by M, and let G' be the bipartite graph obtained from G by the two following operations: Each vertex v of G is turned into two vertices v^s and v^t in G', and each arc (u, v) is turned into the edge $\{u^s, v^t\}$. A loop on a vertex u is simply represented as the edge $\{u^s, u^t\}$. Let A be the symmetric adjacency matrix of G' (when the vertices are ordered $v_0^s, v_1^s, \dots, v_m^s, v_0^t, \dots, v_m^t$).

It is well-known that cycle covers of G and perfect matchings of G' are in one-to-one correspondence. This one-to-one correspondence shows that the determinant of M equals the sum of the weights of the perfect matchings in G'. If a perfect matching in G' is considered as a cycle cover with length-2 cycles, the weight of the cycle cover is the square of the weight of the perfect matching. Indeed, in the cycle cover, all the arcs of the length-2 cycles have to be considered, that is each edge contributes twice to the product. Lemma 7 and the fact that there is no loop in G' show that

$$\det(A) = \sum_\mu w(\mu)^2 = \Big(\sum_\mu w(\mu)\Big)^2,$$

where μ ranges over all perfect matchings of G' and $w(\mu)$ is the weight of the perfect matching μ. The second equality holds as the field has characteristic 2.

Finally, it is shown in [14] that $p = \det(M)$, and we showed that $\det(M) = \sum_\mu w(\mu)$ and $\det(A) = \left(\sum_\mu w(\mu)\right)^2$. Therefore, $\det(A) = \det(M)^2 = p^2$. ◀

This proposition raises the following question: Let f be a family of polynomials such that $f^2 \in \mathsf{VP}_{ws}$. Does f belong to VP_{ws}? This question is discussed with more details in the next section.

3.2 Is the partial permanent complete in characteristic 2?

▶ **Definition 9.** Let $X = (X_{ij})$ be an $(n \times n)$ matrix. The partial permanent of X, as defined by Bürgisser [4], is

$$\mathrm{per}^*(X) = \sum_\pi \prod_{i \in \mathrm{def}(\pi)} X_{i\pi(i)},$$

where the sum ranges over the injective partial maps from $[n] = \{1, \ldots, n\}$ to $[n]$ and $\mathrm{def}(\pi)$ is the domain of the partial map π.

The family (PER_n^*) is the family of polynomials such that PER_n^* is the partial permanent of the $(n \times n)$ matrix whose coefficients are the indeterminates X_{ij}.

▶ **Lemma 10.** *Let G be the complete bipartite graph with two independent sets of vertices V_r and V_c such that the edge between r_i and c_j is labelled by B_{ij} (the matrix B is the biadjacency matrix of G). Then the partial permanent of B is equal to the sum of the weights of the partial matchings of G.*

A partial matching in a graph G is a set of pairs of vertices connected by an edge such that no vertex appears in more than a pair. Equivalently, a partial matching can be seen as a set of disjoint edges. The weight of a partial matching is the product of the weights of its edges.

The proof of the lemma is quite straightforward as a injective partial map π from $[n]$ to $[n]$ exactly defines a partial matching in G such that for $i \in \mathrm{def}(\pi)$, r_i is matched with $c_{\pi(i)}$.

▶ **Lemma 11.** *Let G be the complete bipartite graph with two independent sets of vertices V_r and V_c such that the edge between r_i and c_j is labelled by B_{ij} (the matrix B is the biadjacency matrix of G). Let A be its adjacency matrix. Then in characteristic 2,*

$$\det(A + I_{2n}) = (\mathrm{per}^*(B))^2,$$

where I_{2n} is the identity matrix of size $2n$.

Proof. By Lemma 7, to compute a determinant in characteristic 2, one can focus only on cycles of length at most 2. A cycle cover with such cycles actually is a partial matching when the graph is symmetric (length-2 cycles define the pairs of vertices, and length-1 cycles are isolated vertices). Considering G as a symmetric digraph, the weight of a cycle cover is equal to the product of the weights of its loops and the square of the weights of the edges it uses (a length-2 cycle corresponds to an edge).

Consider the graph G' obtained from G by adding weight-1 loops on all its vertices. In other words, G' is the graph whose adjacency matrix is $A + I_{2n}$. By the previous remark, and by the fact that the loops have weight 1, the determinant of $A + I_{2n}$ is

$$\det(A + I_{2n}) = \sum_\mu w(\mu)^2 = \left(\sum_\mu w(\mu)\right)^2$$

where μ ranges over the partial matchings of G' and $w(\mu)$ is the weight of the partial matching μ. The second equality is true as the characteristic of the field is 2.

Recall now that G is bipartite. Of course, the partial matchings of G and G' are the same. So

$$\mathrm{per}^*(B) = \sum_{\mu} w(\mu),$$

where μ ranges over the partial matchings of G. This proves the lemma. ◀

This lemma shows in particular that for computing the parity of the number of partial matchings in a bipartite graph, it is sufficient to compute a determinant (this is the case where G is not edge-weighted). Therefore, this problem is solvable in polynomial time. This was already mentioned by Valiant [19] but without any proof or reference.

▶ **Theorem 12.** *In characteristic* 2*, the family* $((\mathrm{PER}^*)_n^2)$ *is in* VP_{ws}.

Proof. The previous lemma shows that the polynomial $(\mathrm{PER}^*)_n^2$ is a p-projection of DET_{2n} in characteristic 2. Thus, $((\mathrm{PER}^*)_n^2)$ is in VP_{ws} as $(\mathrm{DET}_n) \in \mathsf{VP}_{ws}$ [14]. ◀

Suppose that (PER_n^*) is VNP-complete. Then every VNP family (f_n) is a p-projection of (PER_n^*), and thus (f_n^2) is a p-projection of $((\mathrm{PER}^*)_n^2)$. Let $\mathsf{VNP}^2 = \{(f_n^2) : (f_n) \in \mathsf{VNP}\}$ be the class of *squares of* VNP *families.* This implies the following corollary of the theorem:

▶ **Corollary 13.** *In any field of characteristic* 2*, if* (PER_n^*) *is* VNP*-complete, then* $\mathsf{VNP}^2 \subseteq \mathsf{VP}_{ws}$.

This situation is unlikely to happen. In particular, it would be interesting to investigate whether this inclusion implies that $\mathsf{VP}_{ws} = \mathsf{VNP}$ in characteristic 2. Let us now give another consequence of (PER_n^*) being VNP-complete. This only holds for finite fields of characteristic 2 but may give a stronger evidence that (PER_n^*) is unlikely to be VNP-complete.

▶ **Theorem 14.** *If the partial permanent family is* VNP*-complete in a finite field of characteristic* 2*, then* $\oplus\mathsf{P/poly} = \mathsf{NC}^2\mathsf{/poly}$*, and the polynomial hierarchy collapses to the second level.*

The proof of this theorem uses the *boolean parts* of Valiant's complexity classes defined in [4]. In the context of finite fields of characteristic 2, the boolean part of a family (f_n) of polynomials with coefficients in the ground field $\mathbb{F}_2$ is the function $bp_f : \{0,1\}^* \to \{0,1\}$ such that for $x \in \{0,1\}^n$, $bp_f(x) = f_n(x) \pmod 2$. The boolean part $\mathrm{BP}(C)$ of a Valiant's class C is the set of boolean parts of all $f \in C$.

Proof. Let (f_n) be a VNP family and (φ_n) its boolean part. As $\varphi_n(x) \in \{0,1\}$ for all $x \in \{0,1\}^n$, (φ_n) is the boolean part of (f_n^2) too. This shows that $\mathrm{BP}(\mathsf{VNP}) \subseteq \mathrm{BP}(\mathsf{VNP}^2)$. By Corollary 13, $\mathsf{VNP}^2 \subseteq \mathsf{VP}_{ws} \subseteq \mathsf{VP}$. Thus, $\mathrm{BP}(\mathsf{VNP}) \subseteq \mathrm{BP}(\mathsf{VNP}^2) \subseteq \mathrm{BP}(\mathsf{VP})$ and as $\mathsf{VP} \subseteq \mathsf{VNP}$

$$\mathrm{BP}(\mathsf{VP}) = \mathrm{BP}(\mathsf{VNP}).$$

Bürgisser [4] shows that in a finite field of characteristic 2, $\oplus\mathsf{P/poly} = \mathrm{BP}(\mathsf{VNP})$, and $\mathrm{BP}(\mathsf{VP}) \subseteq \mathsf{NC}^2\mathsf{/poly}$. Hence, $\oplus\mathsf{P/poly} \subseteq \mathsf{NC}^2\mathsf{/poly}$. Moreover, $\mathsf{NC}^2\mathsf{/poly} \subseteq \mathsf{P/poly} \subseteq \oplus\mathsf{P/poly}$ hence we conclude that

$$\oplus\mathsf{P/poly} = \mathsf{NC}^2\mathsf{/poly}.$$

The collapse of the polynomial hierarchy follows from a non uniform version of the Valiant-Vazirani Theorem [20]: Theorem 4.10 in [4] states that $\mathsf{NP/poly} \subseteq \oplus\mathsf{P/poly}$. Therefore,

$$\mathsf{NC}^2/\mathsf{poly} \subseteq \mathsf{NP/poly} \subseteq \oplus\mathsf{P/poly} = \mathsf{NC}^2/\mathsf{poly}.$$

In particular, $\mathsf{P/poly} = \mathsf{NP/poly}$ and Karp and Lipton [10] showed that this implies the collapse of the polynomial hierarchy to the second level. ◀

4 Conclusion

As was already mentioned, our results can be refined by using a modified version of the skinny size in which multiplications by constants do not count (this is the size considered in [12]). Let us call *green size* this variant. Furthermore, if the polynomial is given as a formula rather than as a weakly-skew circuits, some better bounds can be obtained. These two improvements are detailed in [5, Sections 2 and 3]. Table 1 compares the results obtained, in this paper and in previous ones. The bounds are given for a formula of green size e and for a weakly-skew circuit of green size e with i input gates labelled by a variable, and take into account the improvements explained in the long version.

	Non-symmetric matrix	Symmetric matrix
Formula	$e+1$	$2e+1$[a]
Weakly-skew circuit	$(e+i)+1$	$2(e+i)+1$

[a] The bound is achieved if and only if the entries can be complex numbers. Else, the bound is $2e+2$.

Table 1 Bounds for determinantal representations of formulas and weakly-skew circuits. The bounds for symmetric representations are new, and the bound for a non-symmetric representation of a weakly-skew circuit is a slight improvement of known bounds.

The $(e+1)$ bound for the representation of a formula by a (non-symmetric) matrix determinant was given in [12] by a method purely based on matrices. We show in [5, Section 2.1] that this bound can also be obtained directly from Valiant's original proof [18]. Along the way, we show that Valiant's proof contained a little flaw that was surprisingly never pointed out in the literature (and is present in more recent texts such as [4]). The $(e+i+1)$ bound for the representation of a polynomial computed by a weakly-skew circuit can be obtained from the $(m+1)$ bound (where m is the fat size of the circuit) obtained in [14] if we use our minimization lemma [5, Lemma 15] as well as a similar trick as in the proof of [5, Theorem 5]. Both bounds for the symmetric cases are given in the long version of this paper.

All of these results are valid for any field of characteristic different from 2. We showed that there are some important differences in fields of characteristic 2 for the complexity of polynomials. The open question of characterizing which polynomials can be represented as determinants of symmetric matrices is quite intriguing. Note that a lot of *variants* of the irreprensentable polynomial $xy+z$ (such as $xy+z+xyz+1$ and $xy+1$) do admit symmetric determinantal representations.

References

1 Amos Beimel and Anna Gál. On arithmetic branching programs. *J. Comput. Syst. Sci.*, 59(2):195–220, 1999.

2 Stuart J. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Inform. Process. Letters*, 18:147–150, 1984.

3 P. Brändén. Obstructions to determinantal representability. *ArXiv e-prints*, April 2010. http://adsabs.harvard.edu/abs/2010arXiv1004.1382B.

4 Peter Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory.* Algorithms and Computation in Mathematics. Springer, 2000.

5 Bruno Grenet, Erich L. Kaltofen, Pascal Koiran, and Natacha Portier. Symmetric Determinantal Representation of Formulas and Weakly Skew Circuits. http://arxiv.org/abs/1007.3804v2, 2010.

6 Bruno Grenet, Thierry Monteil, and Stephan Thomassé. Symmetric determinantal representations in characteristic 2. In preparation.

7 J. William Helton, Scott A. McCullough, and Victor Vinnikov. Noncommutative convexity arises from linear matrix inequalities. *J. Funct. Anal.*, 240(1):105–191, November 2006. http://math.ucsd.edu/~helton/osiris/NONCOMMINEQ/convRat.ps.

8 J.W. Helton and V. Vinnikov. Linear matrix inequality representation of sets. *Communications on Pure and Applied Mathematics*, 60(5):654–674, 2006. http://arxiv.org/pdf/math.OC/0306180.

9 Erich Kaltofen and Pascal Koiran. Expressing a fraction of two determinants as a determinant. In David Jeffrey, editor, *ISSAC 2008*, pages 141–146, New York, N. Y., 2008. ACM Press. URL: http://www.math.ncsu.edu/~kaltofen/bibliography/08/KaKoi08.pdf.

10 R.M. Karp and R.J. Lipton. Turing machines that take advice. *L'Enseignement Mathématique*, 28:191–209, 1982.

11 A.S. Lewis, P.A. Parrilo, and M.V. Ramana. The Lax conjecture is true. *Proceedings of the American Mathematical Society*, 133(9):2495–2500, 2005. http://arxiv.org/pdf/math.OC/0304104.

12 H. Liu and K.W. Regan. Improved construction for universality of determinant and permanent. *Inf. Process. Lett.*, 100(6):233–237, 2006.

13 M. Mahajan and V. Vinay. Determinant: Combinatorics, algorithms, and complexity. *Chicago Journal of Theoretical Computer Science*, 5(1997):730–738, 1997.

14 G. Malod and N. Portier. Characterizing Valiant's algebraic complexity classes. *J. Complexity*, 24(1):16–38, 2008. Presented at MFCS'06.

15 Noam Nisan. Lower bounds for non-commutative computation. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 410–418. ACM, 1991.

16 Ronan Quarez. Symmetric determinantal representation of polynomials. http://hal.archives-ouvertes.fr/hal-00275615/en/, April 2008.

17 S. Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE T. Inf. Syst.*, 75(1):116–124, 1992.

18 L. G. Valiant. Completeness classes in algebra. In *Proc. 11th STOC*, pages 249–261, New York, N.Y., 1979. ACM.

19 L.G. Valiant. Completeness for parity problems. *Computing and Combinatorics*, pages 1–8, 2005.

20 L.G. Valiant and V.V. Vazirani. NP is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47:85–93, 1986.

21 J. von zur Gathen. Feasible arithmetic computations: Valiant's hypothesis. *J. Symb. Comput.*, 4(2):137–172, 1987.

Randomness Efficient Testing of Sparse Black Box Identities of Unbounded Degree over the Reals

Markus Bläser[1] and Christian Engels[2]

1 Computer Science, Saarland University
Postfach 151150, 66041 Saarbrücken, Germany
mblaeser@cs.uni-saarland.de

2 Computer Science, Saarland University
Postfach 151150, 66041 Saarbrücken, Germany
engels@cs.uni-saarland.de

Abstract

We construct a hitting set generator for sparse multivariate polynomials over the reals. The seed length of our generator is $\mathrm{O}(\log^2(mn/\epsilon))$ where m is the number of monomials, n is number of variables, and $1-\epsilon$ is the hitting probability. The generator can be evaluated in time polynomial in $\log m$, n, and $\log 1/\epsilon$. This is the first hitting set generator whose seed length is independent of the degree of the polynomial. The seed length of the best generator so far by Klivans and Spielman [16] depends logarithmically on the degree.

From this, we get a randomized algorithm for testing sparse black box polynomial identities over the reals using $\mathrm{O}(\log^2(mn/\epsilon))$ random bits with running time polynomial in $\log m$, n, and $\log \frac{1}{\epsilon}$.

We also design a deterministic test with running time $\tilde{\mathrm{O}}(m^3n^3)$. Here, the $\tilde{\mathrm{O}}$-notation suppresses polylogarithmic factors. The previously best deterministic test by Lipton and Vishnoi [18] has a running time that depends polynomially on $\log \delta$, where δ is the degree of the black box polynomial.

1998 ACM Subject Classification F. Theory of Computation

Keywords and phrases Descartes' rule of signs, polynomial identity testing, sparse polynomials, black box testing

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.555

1 Introduction

Polynomial identity testing is the problem of testing if a polynomial P is equal to zero. Applications include primality testing [1] and testing if a graph has a perfect matching [21], just to mention a few. There are also results in complexity theory which use identity testing as an ingredient. This includes $\mathsf{IP} = \mathsf{PSPACE}$ [23] and the PCP theorem [5, 6].

Of course, if we are given the coefficients of the polynomial, this is an easy problem. The problem gets interesting if the polynomial is given in a compact form, either by a circuit or by a black box. Though these two variants look similar, they are of a very different nature. If we are given an arithmetic circuit C that computes the polynomial P, it is easy to *find* a point ξ with $P(\xi) \neq 0$ provided that $P \not\equiv 0$. For some constant c, the point $\xi = (2^{2^{c\cdot|C|}}, 2^{2^{2\cdot c\cdot|C|}}, \dots, 2^{2^{n\cdot c\cdot|C|}})$ is such a point: First note that the degree d of P and the size of the coefficients of P are bounded by $2^{|C|}$ and $2^{2^{|C|}}$, resp. The Kronecker substitution, which maps each variable X_i to Y^{d^i} for $1 \leq i \leq n$, is an injective mapping of the set of n-variate polynomials of degree $\leq d$ to univariate polynomials such that the degree of the

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 555–566
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

univariate polynomial is bounded by d^{n+1} and the coefficients are preserved. It is easy to see that for a nonzero univariate monic polynomial, every real number larger than the absolute values of all coefficients cannot be a root. The point ξ is obtained by performing a Kronecker substitution and then plugging in a large enough number.

This point ξ is too large to be constructed explicitly in polynomial-time but it can be computed by a polynomial size circuit via repeated squaring. But we do not know how to *evaluate* the circuit at this point. This means that in the circuit model, the polynomial identity testing problem is equivalent to deciding whether a circuit that computes a number computes the value zero, see [3]. There is an efficient randomized algorithm for this problem which chooses a random prime with a polynomial number of bits and evaluates the circuit modulo this prime. In particular, if $\mathsf{RP} = \mathsf{P}$, then there is an efficient deterministic algorithm for this problem. On the other hand, derandomizing this algorithm implies circuit lower bounds [14].

In the black box model, evaluation is of course no problem at all; the black box does it for us. It is even sufficient that the black box only tells us whether the polynomial evaluated at the query point is zero or not. In the black box model, randomization is inherently needed for polynomial-time algorithms (see [16]). Why? First consider a deterministic algorithm. We claim that when the only information that we have is that the polynomial in the black box P has $\leq m$ monomials, then the algorithm has to query the black box at least m times. This is shown by an adversary argument: Every query at a point ξ is answered with zero. Each answer gives a linear equation $P(\xi) = \sum_{\mu=1}^{m} \alpha_\mu \xi_1^{e_{\mu,1}} \cdots \xi_n^{e_{\mu,n}} = 0$ on the coefficients $\alpha_1, \dots, \alpha_m$. As long as less than m queries are done, the system of equations has a nontrivial solution. So the answer to the queries can be produced by two polynomials, the zero polynomial and a nonzero polynomial given by the nontrivial solution above. If now a polynomial-time randomized algorithm would use less than $(1-\epsilon) \cdot \log m$ random bits, then derandomizing it trivially by running over all choices for the random bits will give a deterministic algorithm making less than m queries, which does not exist. (Note that our adversary argument did not make any assumption about the running time of the deterministic algorithm.)

We will focus on the problem in the black box model. Here we are given a black box which we can query at specific points. This black box will evaluate our polynomial at the points and return the value in one time step. For identity testing, it is of course sufficient to know whether the polynomial evaluates at the query point to zero or not. Since we cannot inspect the polynomial other than by querying values, we need a hitting set, that is, a set H such that for every potential polynomial P in the black box, there is a point $x \in H$ with $P(x) \neq 0$ or, stronger, for some fraction of all $x \in H$, $P(x) \neq 0$. A hitting set generator even allows us to sample from a hitting set at random. We will call the number of random bits used by a hitting set generator its seed length. For a formal definition, see Definition 2.4. The so-called Schwartz-Zippel test [22, 24] was one of the first hitting set generators (see [20] for an alternative proof over finite fields). It is designed for dense polynomials and works over arbitrary (large enough) fields. If the polynomial has degrees $\delta_1, \dots, \delta_n$ in the variables $X_1, \dots, X_n$, then the seed length of the generator is $\sum_{i=1}^{n} \lceil \log(\delta_i + 1) \rceil + n \log n$. This seed length was improved by Chen and Kao [13] to $\sum_{i=1}^{n} \lceil \log(\delta_i + 1) \rceil$. However, their test works only for integer coefficients and makes some assumptions on the size of the coefficients. So strictly speaking, their generator is incomparable. Lewin and Vadhan [17] extended the work by Chen and Kao to fields of positive characteristic. Bläser, Hardt, and Steurer [12] constructed a hitting set generator with asymptotically optimal seed length $(1 + \mathrm{o}(1)) \sum_{i=1}^{n} \log(\delta_i + 1)$. Note that a polynomial with degrees $\delta_1, \ldots, \delta_n$ can have $(\delta_1 + 1) \cdots (\delta_n + 1)$ monomials, so we get a matching lower bound on the seed length by the argument outlined above. Prior to this, Agrawal and Biswas [1] achieved the same seed length, but only if the polynomial is given by a circuit.

Klivans and Spielman [16] considered the case of sparse polynomials. This means that the number of monomials is bounded by some parameter m. For polynomials with a bound of δ on the total degree, the running time of their algorithm is polynomial in $\log m$, n, $\log \delta$, and $\log \frac{1}{\epsilon}$. The algorithm needs $\mathrm{O}(\log(mn\delta/\epsilon))$ random bits and has error probability bounded by ϵ. In this paper, we focus on sparse polynomials over the reals. Here Descartes' rule of signs says that a univariate polynomial with at most m monomials has a most $m-1$ positive real roots. This gives an efficient hitting set generator ("plug in a random integer between 1 and $2m$") for univariate polynomials with optimal seed length, which is independent of the degree. This suggests that there should also be hitting set generators for sparse multivariate polynomials over the reals that are independent of the degree. There are multivariate versions of Descartes' rule of signs like Khovanskii's theorem [15] (see also [9, 10] for improvements). However, the size of the resulting hitting set is exponential in the number of variables, which is far too large.

In this work, we give an efficient hitting set generator for sparse polynomials over the reals with running time and seed length independent of the degree of the polynomial. This results in a faster algorithm for real polynomials with high degree but few monomials. In particular, we obtain a randomized algorithm that uses $\mathrm{O}(\log^2(mn/\epsilon)) = \mathrm{O}(\log^2 m + \log^2 n + \log^2 \frac{1}{\epsilon})$ random bits, has running time polynomial in $\log m$, n, and $\log 1/\epsilon$, and error probability $\leq 1/\epsilon$. So compared to the algorithm by Klivans and Spielman, the dependence on $\log m$ and $\log n$ is slightly worse but we are completely independent of the degree δ. We also construct a deterministic algorithm with running time $\tilde{\mathrm{O}}(m^3 n^3)$. Here, the $\tilde{\mathrm{O}}$-notation suppresses polylogarithmic factors. The previously best deterministic test by Lipton and Vishnoi [18] has a running time that depends polynomially on $\log \delta$ (see also [11]).

We conclude by pointing out that the situation over the reals is a special one. Over arbitrary fields, the runtime dependence on the degree is necessary. A short example will show this. Given a field $\mathbb{F}_{p^k}$ for some prime p and a polynomial $P(X) = X^{p^{ck}} - X$, $P(X) \in \mathbb{F}_{p^k}[X]$. The polynomial is not equal to $0 \in \mathbb{F}_{p^k}[X]$; however, it evaluates to zero at every $a \in \mathbb{F}_{p^k}$ and even at every $b \in \mathbb{F}_{p^{ck}}$. Hence, the running time and number random bits will depend on the degree of the polynomial. Otherwise, we could not distinguish $P(X)$ from zero.

2 Hitting Set Generators and Transformations

▶ **Definition 2.1.** We will denote by $\mathbb{R}_m[X_1, \ldots, X_n]$ the set of all polynomials with at most m monomials in the variables $X_1, \ldots, X_n$.

We consider the polynomial identity testing problem restricted to sparse polynomials.

▶ **Definition 2.2.** PIT (n, m) is the problem of deciding if a polynomial in $\mathbb{R}_m[X_1, \ldots, X_n]$ given by a black box is identically zero.

▶ **Definition 2.3.** A set $H \subseteq \mathbb{R}^n$ is a *hitting set* for PIT (n, m) with *hitting probability* $1 - \epsilon$, if for all nonzero $P \in \mathbb{R}_m[X_1, \ldots, X_n]$,

$$\Pr_{x \in H}[P(x) \neq 0] \geq 1 - \epsilon.$$

In the definition above, x is drawn uniformly at random from H. While a hitting set is nice, we also want to be able to efficiently generate elements from the hitting set. Furthermore, the elements in the hitting set should be integers (or from any other subset from $\mathbb{R}$ that can be efficiently represented. But for our purposes, $\mathbb{Z}$ is fine.)

▶ **Definition 2.4.** A function $\mathcal{H} : \{0,1\}^\rho \to \mathbb{Z}^n$ is called a *hitting set generator* for $\mathrm{PIT}\,(n,m)$ with *hitting probability* $1-\epsilon$ and *seed length* ρ if the image of $\mathcal{H}$ is a hitting set with hitting probability $1-\epsilon$. We denote the image of $\mathcal{H}$, that is, the hitting set generated by $\mathcal{H}$, by $\mathrm{im}(\mathcal{H})$.

Of course, we would like to have hitting set generators for $\mathrm{PIT}\,(n,m)$ for each choice of n and m. We call this a uniform hitting set generator. It gets three inputs: n, m, and a seed r of length $\rho(n,m)$. Instead of $\mathcal{H}(n,m,r)$ we will often write $\mathcal{H}_{n,m}(r)$. If we keep n and m fixed and run over all seeds r, we get a hitting set for $\mathrm{PIT}\,(n,m)$. The evaluation time of a hitting set generator is the maximal time needed to compute $\mathcal{H}_{n,m}(r)$ for all $r \in \{0,1\}^{\rho(n,m)}$. The evaluation time is a function of n and m.

▶ **Theorem 2.5.** *If there is a uniform hitting set generator for* $\mathrm{PIT}\,(n,m)$ *with hitting probability* $1-\epsilon > 0$*, seed length* $\rho(n,m)$*, and evaluation time* $t(n,m)$*, then there is*

1. *a deterministic algorithm for* $\mathrm{PIT}\,(n,m)$ *with running time* $\mathrm{O}(2^{\rho(n,m)} \cdot t(n,m))$ *and*
2. *a randomized algorithm for* $\mathrm{PIT}\,(n,m)$ *using* $\rho(n,m)$ *random bits with running time* $\mathrm{O}(t(n,m))$ *and error probability* $1-\epsilon$*.*

Proof. For the deterministic algorithm, we just run over all seeds r, compute $\mathcal{H}_{n,m}(r)$ for each r and check whether the black box polynomial evaluates to zero at $\mathcal{H}_{n,m}(r)$ for all r. If not, P is of course not identically zero. Otherwise, P is identically zero by the definition of hitting set. In the randomized case, we just take a random seed r and evaluate P at $\mathcal{H}_{n,m}(r)$. ◀

▶ **Definition 2.6.** A function $T : \{0,1\}^\rho \times \mathbb{Z}^{n'} \to \mathbb{Z}^n$ is called a *hitting set transformation* from $\mathrm{PIT}\,(n',m')$ to $\mathrm{PIT}\,(n,m)$ with success probability $1-\beta$ if for every hitting set H for $\mathrm{PIT}\,(n',m')$ with hitting probability $1-\epsilon$,

$$\{T(r,x) \mid r \in \{0,1\}^\rho,\ \ x \in H\}$$

is a hitting set with hitting probability $\geq (1-\beta)(1-\epsilon)$.

A hitting set transformation transforms a hitting set for $\mathrm{PIT}\,(n',m')$ into a hitting set for $\mathrm{PIT}\,(n,m)$. It gets an additional random string r of length ρ. The size of the hitting set is extended by a factor of at most 2^ρ and we need an additional ρ random bits to draw a sample from the transformed hitting set. The parameter β measures the loss of “quality”.

Of course, we again want uniform transformations, that is, T gets n and m as additional inputs and $\rho(n,m)$, $n'(n,m)$ and $m'(n,m)$ are functions depending on n and m. There is one algorithm that computes T for all choices of n and m. Again, we will write $T_{n,m}(r,x)$ instead of $T(n,m,r,x)$. For fixed n and m, T is a hitting set transformation in the sense of the definition above. For fixed n and m, the time needed to evaluate a hitting set transformation depends on the size of the elements in the hitting set we apply the transformation to. For a hitting set $H \subseteq \mathbb{Z}^{n'}$, the size of an element $x \in H$ is the bit length of an encoding of x. (We use some standard encoding here, e.g., integers are encoded by a signed binary representation, thus the size is $\log|x| + \mathrm{O}(1)$. The size of tuples of integers is the sum of the sizes of the integers in it.)

▶ **Theorem 2.7.** *Let* $\mathcal{H}_{n',m'} : \{0,1\}^{\rho'} \to \mathbb{Z}^{n'}$ *be a uniform hitting set generator for* $\mathrm{PIT}\,(n',m')$ *with hitting probability* $1-\epsilon$ *the evaluation time of which is bounded by* $t(n',m')$*.*

Let $T_{n,m} : \{0,1\}^\rho \times \mathbb{Z}^{n'} \to \mathbb{Z}^n$ be a uniform hitting set transformation from PIT (n', m') *to* PIT (n,m) *with success probability $1-\beta$ which can be computed in time $g(n,m,s)$ where s is the maximal size of an element in the input hitting set. Then there are*

1. *a deterministic algorithm for* PIT (n,m) *the running time of which is* $\mathrm{O}(2^{\rho(n,m)+\rho'(n',m')} \cdot (g(n,m,t(n',m')) + t(n',m')))$ *and*
2. *a randomized algorithm for* PIT (n,m) *using $\rho(n,m)+\rho'(n',m')$ random bits with running time* $\mathrm{O}(g(n,m,t(n',m')) + t(n',m'))$ *and success probability $(1-\beta)(1-\epsilon)$.*

Above, n' and m' are functions of n and m.

Proof. We start with the randomized algorithm. By the definition of hitting set transformation,

$$\{T_{n,m}(r,x) \mid r \in \{0,1\}^\rho, \;\; x \in \mathrm{im}(\mathcal{H}_{n',m'})\}$$

will be a hitting set for PIT (n,m) with hitting probability $(1-\beta)(1-\epsilon)$. To sample from this set, we choose two seeds $r \in \{0,1\}^{\rho(n,m)}$ and $r' \in \{0,1\}^{\rho'(n',m')}$ uniformly at random. We evaluate P at $T_{n,m}(r, \mathcal{H}_{n',m'}(r'))$ and claim that P is identically zero if the result is zero. Otherwise, P is obviously not identically zero. By the definition of hitting probability, this algorithm succeeds with probability $(1-\beta)(1-\epsilon)$.

For the running time, note that we evaluate $\mathcal{H}_{n',m'}$ once (in time $\mathrm{O}(t(n',m'))$ and $T_{n,m}$ once (in time $\mathrm{O}(g(n,m,t(n',m'))$. Note that the size of $\mathcal{H}_{n',m'}(r')$ can be at most $t(n',m')$.

We get the deterministic algorithm by derandomizing this algorithm in a straight forward manner: Just run over all seeds. ◀

Let $X_1, \dots, X_n$ and $Y_1, \dots, Y_{n'}$ be two sets of variables. A *monomial substitution* σ is a mapping that maps each X_ν to a monomial in $Y_1, \dots, Y_{n'}$. Such a substitution naturally induces a ring homomorphism, which we also call σ, from $\mathbb{R}[X_1, \dots, X_n]$ to $\mathbb{R}[Y_1, \dots, Y_{n'}]$. Since a monomial substitution cannot increase the number of monomials of a polynomial, this ring homomorphism maps polynomials in $\mathbb{R}_m[X_1, \dots, X_n]$ to polynomials in $\mathbb{R}_m[Y_1, \dots, Y_{n'}]$, too. A randomized monomial substitution gets an additional seed $r \in \{0,1\}^\rho$ and maps each X_i to a monomial $\sigma_r(X_i)$ in $Y_1, \dots, Y_{n'}$ that depends on the chosen r.

▶ **Lemma 2.8.** *If there is a randomized monomial substitution as above such that for every nonzero polynomial $P \in \mathbb{R}_m[X_1, \dots, X_n]$,*

$$\Pr_{r \in \{0,1\}^\rho}[\sigma_r(P) \not\equiv 0] \geq 1 - \beta$$

then there is a hitting set transformation from PIT (n', m) *to* PIT (n,m) *with seed length ρ and success probability $1-\beta$.*

Proof. Let H' be a hitting set for PIT (n',m) with hitting probability $1-\epsilon$. We claim that

$$H := \{(\sigma_r(X_1)(y), \dots, \sigma_r(X_n)(y)) \mid r \in \{0,1\}^\rho, \;\; y \in H'\}$$

is a hitting set with hitting probability $(1-\beta)(1-\epsilon)$. Let $P \in \mathbb{R}_m[X_1, \dots, X_n]$ be nonzero. Note that evaluating $P(x)$ for some $x \in H$ is the same as first computing $\sigma_r(P)$ and then plugging in y, where r and y are chosen such that $x = \sigma_r(X_1)(y), \dots, \sigma_r(X_n)(y)$.

With probability $\geq 1-\beta$, $\sigma_r(P)$ is nonzero. In this case, $\sigma_r(P)(y)$ is nonzero with probability $1-\epsilon$. Therefore, H has hitting probability $(1-\beta)(1-\epsilon)$. ◀

The time needed to compute the transformation depends on the degree of the monomials generated by the substitution (and on the time needed to compute the monomials, but this will be negligible in the following). If we have a uniform monomial substitution, that is, a substitution that gets n as an additional parameter and n' depends on n, then we also get a uniform hitting set transformation with $m' = m$.

3 Deterministic Algorithm

We continue with presenting our deterministic algorithm. It will have a running time of $\tilde{\mathrm{O}}\left(m^3n^3\right)$, which is independent of the degree of the polynomial. We start with a simple hitting set generator for $\mathrm{PIT}\,(1,m)$ and build a hitting set transformation from $\mathrm{PIT}\,(1,m)$ to $\mathrm{PIT}\,(n,m)$. The running time of the transformation will depend polynomially on m. Therefore, we only get a good deterministic algorithm with this approach but not a randomized one (which should have a running time that is polynomial in $\log m$).

Algorithm 1 Descartes Generator for $\mathrm{PIT}\,(1,m)$

```
1: Input: Seed $r \in \{0,1\}^{\log(m/\epsilon)}$
2: Use the seed $r$ to choose $y \in \{1,\dots,\frac{m}{\epsilon}\}$ uniformly at random.
3: Output: y
```

▶ **Theorem 3.1.** *Algorithm 1 is a uniform hitting set generator for* $\mathrm{PIT}\,(1,m)$ *with hitting probability* $1-\epsilon$*. The seed length is* $\log\frac{m}{\epsilon}$*, the evaluation time is* $\mathrm{O}(\log\frac{m}{\epsilon})$ *and the output size is* $\log\frac{m}{\epsilon}+\mathrm{O}(1)$*.*

Proof. A real polynomial P with m monomials can have at most $m-1$ positive real roots by Descartes' rule of signs (see e.g., [8]). Hence, the hitting probability is $\geq \frac{m/\epsilon-m}{m/\epsilon} = 1-\epsilon$. The other statements are clear from the construction. ◀

3.1 Hitting Set Transformation

Next, we design a hitting set transformation from $\mathrm{PIT}\,(1,m)$ to $\mathrm{PIT}\,(n,m)$. The construction is based on results by Klivans and Spielman [16, Section 3]. However, the construction by Klivans and Spielman depends on the degree. We refine the construction in such a way that it becomes independent of the degree.

In the following, $x\cdot y$ denotes the standard inner product $\sum_{\nu=1}^{n} x_\nu y_\nu$ of two vectors x and y in an n-dimensional vector space. Let $N=\frac{mn}{\beta}$ and q be a prime with $N<q\leq 2N$. For $1\leq i\leq N$, let a_i denote the vector $\left(1, i \bmod q, i^2 \bmod q, \dots, i^{n-1} \bmod q\right)^T \in \mathbb{Z}_q^n$. The entries of a_i are denoted by $a_{i,1},\dots,a_{i,n}$.

▶ **Lemma 3.2.** *Let* $N=\frac{mn}{\beta}$ *and* q *be a prime such that* $N\leq q\leq 2N$*. Let* $b=(b_1,\dots,b_n)^T\in\mathbb{Z}^n$ *be a vector not equal to zero. Then* $a_i\cdot b$ *is zero for at most* $n-1$ *indices* i*.*

Proof. Let e be the largest exponent such that $q^e|b_\nu$ for every $1\leq\nu\leq n$. Let $c_\nu=\frac{b_\nu}{q^e}$ and c be the vector $(c_1,\dots,c_n)^T$. We now reduce every entry of c modulo q and call this vector $\hat{c}$. This is a nonzero vector by the choice of e.

Let $i_1,\dots,i_n$ be n pairwise distinct indices. Consider the matrix

$$M := \begin{pmatrix} a_{i_1,1} & \cdots & a_{i_1,n} \\ \vdots & \ddots & \vdots \\ a_{i_n,1} & \cdots & a_{i_n,n} \end{pmatrix}.$$

This is a Vandermonde matrix over $\mathbb{Z}_q$. Since $i_1,\dots,i_n$ are pairwise distinct, M is invertible. Hence $M\cdot\hat{c}$ is nonzero as $\hat{c}$ is nonzero. This means that there exists an $\ell\in\{1,\dots,n\}$ such that $a_{i_\ell}\cdot\hat{c}\neq 0$ over $\mathbb{Z}_q$. Therefore $a_{i_\ell}\cdot c\neq 0$ over $\mathbb{Z}$. This implies $a_{i_\ell}\cdot b = a_{i_\ell}\cdot q^e c\neq 0$. Thus, we found at least one i, namely i_ℓ, for which $a_{i_\ell}\cdot b$ is not equal to zero. Since $i_1,\dots,i_n$ were arbitrary, the claim of the lemma follows. ◀

Algorithm 2 Transformation from PIT $(1, m)$ to PIT (n, m)

```
Input: Seed $r \in \{0,1\}^{\log(mn/\beta)}$, $y \in \mathbb{Z}$
Let $N = \frac{mn}{\beta}$ and $N < q \leq 2N$ be a prime.
Use the seed $r$ to choose $i \in \{1, \dots, N\}$ uniformly at random.
Compute $y^{a_{i,\nu}}$ for every $1 \leq \nu \leq n$.
Output: $(y^{a_{i,1}}, y^{a_{i,2}}, \dots, y^{a_{i,n}})$
```

Algorithm 2 is our hitting set transformation. It implements a randomized monomial substitution: We pick one a_i uniformly at random and replace every variable X_ν in our polynomial by $Y^{a_{i,\nu}}$. Our substitution is similar to the one by Klivans and Spielman; however, our choice of the prime q is independent of δ.

▶ **Lemma 3.3** (Correctness). *Algorithm 2 is a uniform hitting set transformation from* PIT $(1, m)$ *to* PIT (n, m) *with success probability* $1 - \beta$.

Proof. Let $P = \sum_{\mu=1}^{m} \alpha_\mu X_1^{\delta_{\mu,1}} \cdots X_n^{\delta_{\mu,n}}$ be a nonzero polynomial in $\mathbb{R}_m[X_1, \dots, X_n]$. Let b_μ be the vector $(\delta_{\mu,1} - \delta_{1,1}, \dots, \delta_{\mu,n} - \delta_{1,1})^T$ for $2 \leq \mu \leq m$. These vectors are nonzero and there are at most $m - 1$ of these vectors.

Let us now look at $a_i \cdot b_\mu$. Lemma 3.2 tells us that for every μ at most $n - 1$ choices of i set this scalar product to zero. Therefore, for at most $(m - 1)(n - 1)$ indices i, there is a μ with $a_i \cdot b_\mu = 0$. There are N possible values for i.

For a randomly chosen i the probability that $a_i \cdot b_\mu \neq 0$ for all μ is at least

$$\frac{N - (m-1)(n-1)}{N} \geq 1 - \beta.$$

So with probability $\geq 1 - \beta$, $a_i \cdot b_\mu \neq 0$ for $2 \leq \mu \leq m$. Let us denote by δ_μ the vector $(\delta_{\mu,1}, \dots, \delta_{\mu,n})^T$. By the definition of b_μ, $a_i \cdot \delta_\mu \neq a_i \cdot \delta_1$ for $2 \leq \mu \leq m$. This means that our monomial $\alpha_1 X_1^{\delta_{1,1}} \cdots X_n^{\delta_{1,n}}$ is not canceled by $\alpha_\mu X_1^{\delta_{\mu,1}} \cdots X_n^{\delta_{\mu,n}}$ for every $2 \leq \mu \leq m$. This implies that the image $\hat{P}$ of P under the monomial substitution $X_\nu \mapsto Y^{a_{i,\nu}}$, $1 \leq \nu \leq n$, is nonzero with probability $1 - \beta$. Hence, the algorithm computes a randomized monomial substitution. Lemma 2.8 finishes the proof. ◀

▶ **Lemma 3.4** (Runtime and Randomness). *Algorithm 2 has a seed length of* $\log \frac{mn}{\beta}$ *and a running time of* $\mathrm{O}\left(\frac{mn}{\beta} \log^2 \frac{mn}{\beta} \log\log \frac{mn}{\beta} + \frac{mn^2}{\beta} \log y\right)$, *where y is the input.*

Proof. Let us start with the runtime. We need $\mathrm{O}(N \log^2 N \log\log N)$ bit operations to find all primes from 2 to $2N$ by the Sieve of Eratosthenes (see e.g., [7]). Hence, we need $\mathrm{O}(\frac{mn}{\beta} \log^2 \frac{mn}{\beta} \log\log \frac{mn}{\beta})$ steps for finding our prime number.

A prime q with $N < q \leq 2N$ exists by Bertrand's postulate. The entries of one a_i have at most $\log(\frac{mn}{\beta}) + \mathrm{O}(1)$ bits. To compute the entries, we need to multiply two numbers of at most $\log(\frac{mn}{\beta}) + \mathrm{O}(1)$ bits because we compute modulo q. We do n multiplications of this form. Hence the computation of one vector a_i takes at most $\mathrm{O}(n \log^2 \frac{mn}{\beta})$ steps. The values $y^{a_{i,\nu}}$ are numbers with $\frac{mn}{\beta} \log y$ bits. By using repeated squaring, we can compute one of the entries in time $\mathrm{O}(\frac{mn}{\beta} \log y)$ and all values $y^{a_{i,\nu}}$, $1 \leq \nu \leq n$, in time $\mathrm{O}(n \cdot \frac{mn}{\beta} \log y)$.

The number of random bits used is clear from the construction. ◀

3.2 Final Algorithm

We now have all the necessary pieces to construct our deterministic algorithm.

▶ **Theorem 3.5.** *There exists a deterministic algorithm which tests if a polynomial given as a black box is equivalent to the zero polynomial in* $\tilde{\mathrm{O}}\left(m^3n^3\right)$ *steps.*

Proof. We take Algorithm 1 and Algorithm 2 to construct a deterministic algorithm for PIT (n, m) using Theorem 2.7. Choose arbitrary ϵ, β, both smaller than $\frac{1}{4}$. Plugging theses values into Theorem 2.7 yields an algorithm with a running time of

$$\mathrm{O}\left(2^{\log mn+\log m}\cdot\left(mn\log^2 mn\log\log mn+mn^2\log m+\log m\right)\right)=\tilde{\mathrm{O}}\left(m^3n^3\right).$$

Note that $t\left(1,m\right)$ is $\log m$. ◀

This construction gives us an efficient deterministic algorithm which has a runtime independent of the degree.

4 Randomized Algorithm

Let us continue with our randomized algorithm. Instead of starting with univariate polynomials, as in the deterministic case, we start with multivariate polynomials that have a significantly smaller number of variables, namely $\lceil\log q\rceil+1$. Note that $\log q$ is roughly $\log m+\log n$.

Our transformation from PIT $(\lceil\log q\rceil+1, m)$ to PIT (n, m) will be very helpful as the number of random bits that the hitting set generator uses is linear in the number of variables. With the transformation mentioned above, we can efficiently reduce the number of variables our underlying hitting set generator has to work on.

Again, we divide the construction into three parts. First, we present a hitting set generator for multivariate polynomials. We continue with our transformation which uses the vectors a_i defined in the previous section. Finally, we combine the generator and the transformation yielding a hitting set generator with runtime polynomial in n, $\log m$ and $\log\frac{1}{\epsilon}$ and seed length $\mathrm{O}(\log^2\frac{mn}{\epsilon})$.

4.1 Hitting Set Generator for Multiple Variables

Our hitting set generator is a modified version of the Schwartz-Zippel generator. We adapt it to sparse polynomials over the reals in such a way that the runtime and the seed length become independent of the degree.

▶ **Lemma 4.1.** *Let* $P\in\mathbb{R}_m[X_1,\dots,X_n]$ *be a nonzero polynomial. Let* z_ν, $1\le\nu\le n$, *be drawn independently and uniformly at random from* $Z\subseteq\mathbb{Z}$ *and let* $z=(z_1,\dots,z_n)$. *Then*

$$\Pr_{z\in Z^n}\left[P\left(z\right)=0\right]\le\frac{mn}{|Z|}.$$

Proof. We prove the lemma by induction in n, in a similar fashion to the proof of the original lemma. If $n=1$ then the claim follows from Descartes' rule of signs.

If $n>1$, we can write P as $\sum_{\mu=1}^{m'}X_1^{\delta_{\mu,1}}P_\mu\left(X_2,\dots,X_n\right)$ where $m'\le m$. We look at the polynomial as a polynomial in $\mathbb{R}[X_2,\dots,X_n][X_1]$, a polynomial in X_1 with coefficients from $\mathbb{R}[X_2,\dots,X_n]$.

We have

$$\begin{aligned}\Pr_z\left[P(z)=0\right] &= \Pr_z\left[P(z_1,\dots,z_n)=0|P_1(z_2,\dots,z_n)=0\right]\cdot\Pr_z\left[P_1(z_2,\dots,z_n)=0\right]\\ &\quad+\Pr_z\left[P(z_1,\dots,z_n)=0|P_1(z_2,\dots,z_n)\neq 0\right]\cdot\Pr_z\left[P_1(z_2,\dots,z_n)\neq 0\right]\\ &\le \Pr_z\left[P_1(z_2,\dots,z_n)=0\right] \qquad (1)\\ &\quad+\Pr_z\left[P(z_1,\dots,z_n)=0|P_1(z_2,\dots,z_n)\neq 0\right]. \qquad (2)\end{aligned}$$

We can bound (1) using the induction hypothesis and (2) by Descartes' rule of signs, as P, after plugging in $z_2,\dots,z_n$, is a nonzero univariate polynomial with at most m monomials. This yields the bound

$$\Pr_z\left[P(z)=0\right]\le\frac{(n-1)m}{|Z|}+\frac{m}{|Z|}=\frac{nm}{|Z|},$$

which proves the lemma. ◀

Algorithm 3 Schwartz-Zippel Hitting Set Generator

```
Input: Seed $r \in \{0,1\}^{n\log(mn/\epsilon)}$
Use the seed $r$ to choose $z \in \{1,\dots,\frac{mn}{\epsilon}\}^n$
Output: $z$
```

▶ **Theorem 4.2.** *Algorithm 3 is a uniform hitting set generator for* PIT (n,m) *with hitting probability* $1-\epsilon$*. The seed length is* $n\log\frac{mn}{\epsilon}$*, the evaluation time is* $\mathrm{O}\left(n\log\frac{mn}{\epsilon}\right)$ *and the output size is* $\mathrm{O}\left(n\log\frac{mn}{\epsilon}\right)$*.*

Proof. The running time and randomness used are clear from the construction. We can use Lemma 4.1 for bounding the error probability: For nonzero P,

$$\Pr_z\left[P(z)=0\right]\le\frac{mn}{|Z|}$$

which in our case can be bounded by $\frac{mn}{mn/\epsilon}=\epsilon$. ◀

As it is, the Schwartz-Zippel hitting set generator is very inefficient. The number of random bits used depends linearly on the number of variables. The reason for this is that in the proof, we assume that P_1 has m monomials but also P as a univariate polynomial in X_1 has m monomials. If we knew tighter bounds then we would be able to reduce the number of random bits used. However, we cannot assume that we know such bounds in the black box model. Therefore, we will try to reduce the number of variables instead by a suitable monomial substitution.

4.2 Hitting Set Transformation

We will use our a_i as previously defined. Let $a_i=(a_{i,1},\dots,a_{i,n})$, N, and q be as in Lemma 3.2. We define $a_{i,\nu,\kappa}$ by

$$a_{i,\nu}=\sum_{\kappa=0}^{\lceil\log q\rceil}a_{i,\nu,\kappa}2^\kappa,$$

that is, $a_{i,\nu,\kappa}$, $0\le\kappa\le s$ is the binary expansion of $a_{i,\nu}$. For the rest of the paper let $s=\lceil\log q\rceil$.

Algorithm 4 Transformation from PIT $(s+1, m)$ to PIT (n, m)

```
**Input:** Seed $r \in \{0,1\}^{\log(mn/\beta)}$, $y_0, \ldots, y_s$
Let $N = \frac{mn}{\beta}$ and $N < q \leq 2N$ be a prime.
Use the seed $r$ to choose $i \in \{1, \ldots, N\}$ uniformly at random.
Set $z_\nu$ to $y_0^{a_{i,\nu,0}} \cdots y_s^{a_{i,\nu,s}}$ for every $1 \leq \nu \leq n$.
**Output:** $(z_1, \ldots, z_n)$
```

▶ **Lemma 4.3** (Correctness). *Algorithm 4 is a uniform hitting set transformation from* PIT $(s+1, m)$ *to* PIT (n, m) *with success probability* $1 - \beta$.

Proof. Let T be our transformation and let P be a nonzero polynomial. T essentially implements the following monomial substitution: $X_\nu \mapsto Y_0^{a_{i,\nu,0}} \cdots Y_s^{a_{i,\nu,s}}$, $1 \leq \nu \leq n$. If we now replace each Y_j by Y^{2^j}, then $X_\nu \mapsto Y^{\sum_{j=1}^{s} a_{i,\nu,j} 2^j} = Y^{a_{i,\nu}}$, $1 \leq \nu \leq n$ and we get the same substitution as used in the transformation for the deterministic algorithm in Section 4.1. Since for this combined substitution, the probability that P is mapped to zero is at most β, this has to be true for the substitution $X_\nu \mapsto Y_0^{a_{i,\nu,0}} \cdots Y_s^{a_{i,\nu,s}}$, too. Now the claim follows from Lemma 2.8. ◀

Before we prove the bounds on our transformation we need to take a short excursion on finding prime numbers. We state a proof of this well-known result for the sake of completeness.

▶ **Lemma 4.4** (Finding primes with few random bits). *We can find a prime q of size $N < q \leq 2N$ with success probability $1 - \epsilon$ in time* poly $\left(\log N, \log \frac{1}{\epsilon}\right)$ *steps using* $\mathrm{O}\left(\log N + \log \frac{1}{\epsilon}\right)$ *random bits.*

Proof. We construct $\log^2 N$ pairwise independent bit strings of length $\log N$. We can do this using only $2 \log N$ random bits as stated by Luby and Wigderson [19]. We deterministically test whether each number is prime by using the algorithm developed by Agrawal, Kayal and Saxena [2] using $\mathrm{O}(\log^6 N)$ steps. By the Chebyshev bound, one of these number is prime with probability $1 - \mathrm{o}(1)$.

We can increase the probability to $1 - \epsilon$ (think of ϵ being a small function) by doing a random walk on an expander graph which costs us $\mathrm{O}\left(\log \frac{1}{\epsilon}\right)$ extra random bits [4]. ◀

▶ **Lemma 4.5** (Runtime and Randomness). *Our hitting set transformation has a seed length of* $\mathrm{O}(\log \frac{mn}{\beta})$ *bits and a runtime polynomial in* $\log m$, n, $\log \frac{1}{\beta}$, *and* $\log y$, *where* $y = \max\{|y_0|, \ldots, |y_s|\}$

Proof. If we want to have an overall error probability of β we have to set the error for the prime finding algorithm to $\beta/2$ and adjust our choice of β in the transformation to $\beta/2$.

Let us take a closer look at the runtime bounds. We need

$$\mathrm{O}\left(\log^8 \frac{2mn}{\beta} + \log \frac{2}{\beta}\right) = \mathrm{O}\left(\log^8 \frac{mn}{\beta}\right)$$

steps for finding the prime number. Computing the a_i takes us, as seen in the previous transformation, $\mathrm{O}(n \log^2 \frac{2mn}{\beta})$ steps. To calculate the z_ν, we multiply at most $s+1$ numbers with $\log y$ bits. This needs total time of $\mathrm{O}(n \cdot s \cdot \log y)$. Since $s = \mathrm{O}(\log m + \log n + \log \frac{1}{\beta})$, the total running time is $\mathrm{poly}(\log m, n, \log \frac{1}{\beta}, \log y)$.

The randomness used is clear. We need $\mathrm{O}(\log \frac{mn}{\beta})$ random bits for generating the a_i as in Lemma 3.4. Our prime finding needs $\mathrm{O}(\log \frac{mn}{\beta})$ random bits as well. ◀

4.3 Final Algorithm

Again, we combine our hitting set generator with our hitting set transformation to get our randomized algorithm.

▶ **Theorem 4.6.** *There exists a probabilistic algorithm for* PIT (n, m)*. It has success probability* $\geq 1 - \epsilon$*, uses* $\mathrm{O}(\log^2 \frac{mn}{\epsilon})$ *random bits, and has runtime polynomial in* n*,* $\log m$ *and* $\log \frac{1}{\epsilon}$*.*

Proof. We combine Algorithm 3 with Algorithm 4 and use again Theorem 2.7. We set both error parameters to be $\frac{\epsilon}{2}$. Let us look at the running time. Note that Algorithm 3 now works on $s + 1$ variables which is $\lceil \log q \rceil + 1$. This is of course in $\mathrm{O}\left(\log \frac{mn}{\epsilon}\right)$. The total running time is

$$\mathrm{poly}\left(\log m, n, \log \frac{1}{\epsilon}, s, \log y\right) + \mathrm{O}(s \log \frac{ms}{\epsilon}).$$

In our case $\log y$ is at most $\log N$. This gives us a runtime bound of $\mathrm{poly}\left(\log m, n, \log \frac{1}{\epsilon}\right)$.

The algorithm uses

$$\mathrm{O}\left((s+1) \log \frac{2m(s+1)}{\epsilon} + \log \frac{mn}{\epsilon}\right)$$

random bits. It is easy to see that this $\mathrm{O}(\log^2 \frac{mn}{\epsilon})$, as $s = \mathrm{O}(\log \frac{mn}{\epsilon})$. ◀

Note that our algorithm also allows for a so-called time-randomness tradeoff like some of the previous tests mentioned in the introduction do. When $\epsilon > \frac{1}{mn}$, then we just run the algorithm with some constant error probability and decrease the error probability to $\frac{1}{mn}$ by doing a random walk on an expander with an additional $\mathrm{O}(\log mn)$ random bits. When $\epsilon \leq \frac{1}{mn}$ then increase the success probability by just spending more time instead of more random bits by just running over all choices for the extra $\mathrm{O}(\log^2 \frac{1}{\epsilon})$ random bits and doing a majority vote. The running time in the second case is only quasipolynomial in $\frac{1}{\epsilon}$, however.

It remains an open problem whether we can bring down the number of random bits to $\mathrm{O}(\log m)$, which would match the lower bound of $(1 - \epsilon) \cdot \log m$. One approach could be to decrease the number of random bits used in the Schwartz-Zippel Hitting Set Generator. While it is clear that we do an overapproximation on the number of monomials for many polynomials in the proof, it is not clear how we can use this fact.

References

1 Manindra Agrawal and Somenath Biswas. Primality and identity testing via chinese remaindering. In *FOCS*, pages 202–209, 1999.

2 Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in P. *Ann. of Math*, 2:781–793, 2002.

3 Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 39(3):1987–2006, 2009.

4 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 2009.

5 Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

6 Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

7 Eric Bach and Jeffrey Shallit. *Algorithmic number theory. Vol. 1.* Foundations of Computing Series. MIT Press, Cambridge, MA, 1996. Efficient algorithms.

8 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in real algebraic geometry*, volume 10 of *Algorithms and computation in mathematics*. Springer, 2003.

9 Frédéric Bihan, J. Maurice Rojas, and Frank Sottile. On the sharpness of fewnomial bounds and the number of components of fewnomial hypersurfaces. In *Algorithms in algebraic geometry*, volume 146 of *IMA Vol. Math. Appl.*, pages 15–20. Springer, New York, 2008.

10 Frédéric Bihan and Frank Sottile. New fewnomial upper bounds from Gale dual polynomial systems. *Mosc. Math. J.*, 7(3):387–407, 573, 2007.

11 Markus Bläser, Moritz Hardt, Richard J. Lipton, and Nisheeth K. Vishnoi. Deterministically testing sparse polynomial identities of unbounded degree. *Inf. Process. Lett.*, 109(3):187–192, 2009.

12 Markus Bläser, Moritz Hardt, and David Steurer. Asymptotically optimal hitting sets against polynomials. 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 345–356. Springer, 2008.

13 Zhi-Zhong Chen and Ming-Yang Kao. Reducing randomness via irrational numbers. In *STOC*, pages 200–209, 1997.

14 Valentine Kabanets and Russell Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. In *STOC*, pages 355–364. ACM, 2003.

15 Askold Khovanskii. *Fewnomials.* AMS press, 1991.

16 Adam Klivans and Daniel A. Spielman. Randomness efficient identity testing of multivariate polynomials. In *STOC*, pages 216–223, 2001.

17 Daniel Lewin and Salil P. Vadhan. Checking polynomial identities over any field: Towards a derandomization? In *STOC*, pages 438–447, 1998.

18 Richard J. Lipton and Nisheeth K. Vishnoi. Deterministic identity testing for multivariate polynomials. In *SODA*, pages 756–760, 2003.

19 Michael Luby and Avi Wigderson. Pairwise independence and derandomization. *Foundations and Trends in Theoretical Computer Science*, 1(4), 2005.

20 Dana Moshkovitz. An alternative proof of the Schwartz-Zippel lemma. Technical report, The Electronic Colloquium on Computational Complexity, 2010.

21 Ketan Mulmuley, Umesh V. Vazirani, and Vijay V. Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7(1):105–113, 1987.

22 Jacob T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.

23 Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.

24 Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *EUROSAM*, volume 72 of *Lecture Notes in Computer Science*, pages 216–226. Springer, 1979.

On Isomorphism Testing of Groups with Normal Hall Subgroups*

Youming Qiao[1], Jayalal Sarma M.N.[2], and Bangsheng Tang[1]

1 **Institute for Theoretical Computer Science,**
Tsinghua University, Beijing, China
jimmyqiao86,megatang@gmail.com

2 **Department of Computer Science & Engineering,**
Indian Institute of Technology Madras, Chennai, India
jayalal@cse.iitm.ac.in

Abstract

A normal Hall subgroup N of a group G is a normal subgroup with its order coprime with its index. Schur-Zassenhaus theorem states that every normal Hall subgroup has a complement subgroup, that is a set of coset representatives H which also forms a subgroup of G. In this paper, we present a framework to test isomorphism of groups with at least one normal Hall subgroup, when groups are given as multiplication tables. To establish the framework, we first observe that a proof of Schur-Zassenhaus theorem is constructive, and formulate a necessary and sufficient condition for testing isomorphism in terms of the associated actions of the semidirect products, and isomorphisms of the normal parts and complement parts.

We then focus on the case when the normal subgroup is abelian. Utilizing basic facts of representation theory of finite groups and a technique by Le Gall in [9], we first get an efficient isomorphism testing algorithm when the complement has bounded number of generators. For the case when the complement subgroup is elementary abelian, which does not necessarily have bounded number of generators, we obtain a polynomial time isomorphism testing algorithm by reducing to generalized code isomorphism problem. A solution to the latter can be obtained by a mild extension of the singly exponential (in the number of coordinates) time algorithm for code isomorphism problem developed recently by Babai in [3]. Enroute to obtaining the above reduction, we study the following computational problem in representation theory of finite groups: given two representations ρ and τ of a group H over $\mathbb{Z}_p^d$, p a prime, determine if there exists an automorphism $\phi : H \to H$, such that the induced representation $\rho_\phi = \rho \circ \phi$ and τ are equivalent, in time poly$(|H|, p^d)$.

Keywords and phrases Group Isomorphism Problem, Normal Hall Subgroups, Computational Complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.567

1 Introduction

The Group Isomorphism problem(GpI) is a computational problem intriguing for both complexity theorists as well as computational group theorists. Given two finite groups G and H, the problem asks to test if they are isomorphic, that is the existence of a bijection $\phi : G \to H$

[*] Part of this work was done while the second author was a postdoctoral fellow at the Institute for Theoretical Computer Science at Tsinghua University, Beijing, China. All the authors are supported in part by the National Natural Science Foundation of China Grant 60553001, 61073174, 61033001 and the National Basic Research Program of China Grant 2007CB807900, 2007CB807901.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 567–578
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

preserving group operations, namely $\forall g, h \in G$, $\phi(g \cdot h) = \phi(g) \cdot \phi(h)$. Naturally, the complexity of the problem depends on how the group is represented: if the groups are given as presentations (generators and relations), then it is undecidable [8, 1]. For permutation groups given as generators, the best upper bound known [6] is PSPACE.

The least succinct input format, multiplication table (Cayley table), gives rise to a more interesting scenario from a complexity theoretic perspective. For this case, the problem is known to be easier than the well-known Graph Isomorphism problem (GRI) [13], thus giving an upper bound of NP ∩ coAM. However, unlike many other isomorphism-type problems, a reduction in the reverse direction is not known[13]. A recent work [7] shows that GRI can not be AC^0 reducible to GPI. Another distinction between GPI and GRI lies in the best known algorithms for them. The best known algorithm for GRI is $2^{\tilde{O}(\sqrt{n})}$ [5], where n is the size of the graph. For groups of size n with b generators, in [16] Tarjan is credited for pointing out an $n^{b+O(1)}$ algorithm. Then by the observation that every group has a generating set of size $\lceil \log n \rceil$, we get an $n^{\log n + O(1)}$ algorithm for testing isomorphism of general groups. This is improved by Lipton, Snyder and Zalcstein [14], who gave an algorithm running in $O(\log^2 n)$ space. However, whether a polynomial time algorithm exists is still open.

1.1 Progress for testing isomorphism of restricted classes of groups

There has been some progress on group isomorphism problem for restricted classes of groups. The class of groups with bounded number of generators (say, of size b) can be tested efficiently by the $n^{b+O(1)}$ algorithm. For abelian groups, Savage [19] first gave an $O(n^2)$ algorithm, which was improved to $O(n \log n)$ by Vikas [24] and finally to $O(n)$ by Kavitha [11]. Little is known beyond abelian groups until 2008, when Le Gall [9] showed that isomorphism of groups in the form of semidirect products of an abelian group and a cyclic group, whose orders are coprime, can be tested in almost linear time even in the model of black-box groups. The class of p-groups seems to be the current barrier, though recent works by Wilson [25, 26] on the structure of p-groups are noteworthy.

Recently, Kayal and Nezhmetdinov [12] and Wilson [27] address the problem of finding the factors of a group under the direct product operation (Wilson [27] considers a stronger model, that is permutation groups given as generators). They show that given a group, all its direct factors can be computed efficiently. As pointed out in [12], this result can be interpreted in the context of isomorphism testing as follows: by Remak-Krull-Schmidt theorem, two groups are isomorphic if and only if their direct factors are isomorphic up to appropriate correspondence of the factors. Thus, the class of groups that are direct products of groups with known efficient isomorphism testing procedure can be tested efficiently.

This argument suggests the following strategy: suppose for some group class, the groups can be decomposed into smaller subgroups in some canonical way. Then after decomposition, isomorphism testing of the original groups may reduce to testing isomorphism of the building blocks, and then pasting solutions of building blocks back together. In the case of direct product, decomposition is solved in [12] and [27], and "pasting" is trivial due to Remak-Krull-Schmidt theorem. Now it is natural to ask if this strategy can be extended to the case of less stringently defined products. The next natural target is that of semidirect products, which is already considered in [9]. A group G is the semidirect product of a normal subgroup N by a subgroup H if $G = NH$ and $N \cap H = \{\mathrm{id}\}$. Every $h \in H$ can act on N by conjugation, giving rise to a homomorphism from H to $\mathrm{Aut}(N)$, called the action associated with the semidirect product. Unlike direct product, a semidirect product $G = N \rtimes_\tau H$ is canonical only with respect to the associated action. For the special class considered in [9], due to this reason Le Gall needs to solve the problem of testing whether two automorphisms of abelian groups are conjugate or not (when the automorphisms satisfy some property), for which he gives an efficient algorithm.

1.2 Our result: a framework for testing isomorphism of groups with normal Hall subgroups

A Hall divisor m of an integer n is a divisor of n such that $(m, n/m) = 1$. A normal Hall subgroup is a normal subgroup whose order is a Hall divisor of the order of the group. In this paper, we consider the class of groups with at least one normal Hall subgroup, and use $\mathcal{H}$ to denote this group class. It turns out this condition suggests some interesting properties of the group structure. For a given Hall divisor of the size of the group, if the normal Hall subgroup of this size exists then it is a characteristic subgroup. Schur-Zassenhaus theorem states that a normal Hall subgroup always has a complement, that is a set of representatives forming a subgroup. Thus the semidirect product arises naturally for groups in $\mathcal{H}$. Note that $\mathcal{H}$ contains all groups of order $2 \cdot p^k$, p a prime other than 2, and all nilpotent groups that are not p-groups. To see the first point, note that a Sylow p-subgroup is normal as it is of index 2, and the second point follows due to that a nilpotent group is direct product of its Sylow subgroups.

Inspired by [9], we begin with formalizing the strategy for isomorphism testing discussed in Section 1.1 for the class $\mathcal{H}$. As a first step, we need to have an efficient decomposition procedure. The observation is that the proof of Schur-Zassenhaus theorem is efficiently constructive, establishing the following theorem about finding a complement of a normal Hall subgroup.

▶ **Theorem 1.1.** (Algorithmic Schur-Zassenhaus theorem) *For a group G of order n, given as multiplication table, all its normal Hall subgroups can be computed in time $O(n^4)$. Given a specific normal Hall subgroup, one of its complements can be computed in time $O(n^4)$.*

In the second step, we need to consider how isomorphism of the original groups connects isomorphisms of the components. Our next result, which has been discovered by Taunt [23] in the context of construction of finite groups, is the formulation of a necessary and sufficient condition of the original groups being isomorphic in Theorem 4.1. That condition involves the actions associated with the semidirect products, and the isomorphisms of the normal and complement parts. It is not listed here, partly due to its technicality, but the main reason is that as discussed, we need to turn our focus to the case when the factors of semidirect product are efficiently testable. The following notations will help us to talk about the group classes of the factors in the semidirect product. Given two groups X and Y whose orders are coprime, $\mathcal{H}(X, Y)$ is the class of groups with a normal Hall subgroup isomorphic with X, and a complement isomorphic with Y. For two group classes $\mathcal{X}$ and $\mathcal{Y}$, $\mathcal{H}(\mathcal{X}, \mathcal{Y})$ is the class of groups with a normal Hall subgroup X from $\mathcal{X}$ and the complement Y from $\mathcal{Y}$. Note that X being a Hall subgroup implies that the orders of X and Y are coprime. That is $\mathcal{H}(\mathcal{X}, \mathcal{Y}) = \bigcup_{X \in \mathcal{X}, Y \in \mathcal{Y}, \gcd(|X|,|Y|)=1} \mathcal{H}(X, Y)$.

We set notations for some group classes with known isomorphism testing/computing procedure. Let $\mathcal{A}$ be the class of abelian groups. As subclasses of $\mathcal{A}$, $\mathcal{A}_p$ is the class of abelian p-groups, and $\mathcal{E}$ is the class of elementary abelian groups. $\prod \mathcal{E}$ is the class of direct products of elementary abelian groups. $\mathcal{B}_b$ is the class of groups with the number of generators bounded by b. Note that $\mathcal{B}_2$ includes all finite simple groups[1], symmetric

[1] For readers unfamiliar with this fact, c.f. the first theorem in [15], and note that a simple abelian group must be a cyclic group with prime order.

groups and cyclic groups. When the specific number of generators is not of our concern, we will simply write $\mathcal{B}$. $\mathcal{C} = \mathcal{B}_1$ is the class of cyclic groups. Finally, we let $\mathcal{K}$ be a variable taking values from the class of groups with known efficient isomorphism testing/computing procedure. In this article, we mainly consider the case when $\mathcal{K}$ is $\mathcal{A}$ or $\mathcal{B}$, or subclasses of $\mathcal{A}$ or $\mathcal{B}$. To give an example of the use of the notations, the main result of [9] is an efficient isomorphism testing/computing algorithm of $\mathcal{H}(\mathcal{A}, \mathcal{C})$, while our main concrete results are efficient algorithms for $\mathcal{H}(\mathcal{A}, \mathcal{B})$ (when the complement has bounded number of generators), and $\mathcal{H}(\mathcal{A}, \mathcal{E})$ (when the complement is elementary abelian). $\mathcal{H}(\mathcal{A}, \mathcal{B})$ improves the class $\mathcal{H}(\mathcal{A}, \mathcal{C})$ studied in [9].

1.3 Our result: efficient isomorphism testing of $\mathcal{H}(\mathcal{A}, \mathcal{E})$, $\mathcal{H}(\mathcal{A}, \mathcal{B})$

Representation theory of finite groups studies the homomorphisms from abstract groups to general linear groups. Such a homomorphism is called a representation. In Theorem 4.1, when the normal subgroup is an elementary abelian group $\mathbb{Z}_p^k$, p a prime, it naturally gives rise to the following algorithmic problem in representation theory of finite groups which may be of independent interest, which we call AUTOINDUCEDREPEQUIV (short for finding the AUTOmorphism INDUCED REPresentation EQUIValence).

▶ Problem 1. (AUTOINDUCEDREPEQUIV) Given two representations ρ and τ of a group H over $\mathbb{Z}_p^d$, p a prime, determine if there exists an automorphism $\phi : H \to H$, such that the induced representation $\rho_\phi = \rho \circ \phi$ and τ are equivalent, in time poly$(|H|, p^d)$.

The following theorem suggests that AUTOINDUCEDREPEQUIV can not be got around in order to solve isomorphism of groups from $\mathcal{H}(\mathcal{E}, \mathcal{K})$.

▶ **Theorem 1.2.** *For groups from $\mathcal{H}(\mathcal{E}, \mathcal{K})$, isomorphism testing is many-one equivalent to* AUTOINDUCEDREPEQUIV.

Using basic facts from representation theory, it is not hard to solve AUTOINDUCEDREPEQUIV when the number of generators is bounded, giving an efficient testing algorithm of $\mathcal{H}(\mathcal{E}, \mathcal{B})$. The non-trivial case is when the number of generators is not bounded. When the complement is an elementary abelian group, we further reduce AUTOINDUCEDREPEQUIV to a mild generalization[2] of the linear code isomorphism problem in singly exponential time, which asks whether two linear subspaces are the same up to permutation of coordinates in time exponential to the number of coordinates.

▶ **Theorem 1.3.** *For groups from $\mathcal{H}(\mathcal{E}, \mathcal{E})$,* AUTOINDUCEDREPEQUIV *reduces to generalized code isomorphism problem.*

In a recent work [3], Babai presents an algorithm solving the code isomorphism problem in singly exponential time in the number of coordinates, which is logarithmic of the size of the group in our case, allowing us to establish the following.

▶ **Corollary 1.4.** *There is an $O(n^6)$ algorithm testing isomorphism of groups from $\mathcal{H}(\mathcal{E}, \mathcal{E})$.*

It is worth noting that the number of groups in this class is lower bounded by $n^{\Omega(\log n)}$, for certain infinite sequence of group size n. Applying a technique in [9], we extend this further to provide an efficient isomorphism testing of groups from $\mathcal{H}(\mathcal{A}, \mathcal{E})$. An $O(n^{b+5})$ algorithm for $\mathcal{H}(\mathcal{A}, \mathcal{B}_b)$ can also be derived in this framework, rediscovering what is known in Section 8.9, [10] (see Section 4.2).

[2] See Section 5 for specific points of generalization.

▶ **Theorem 1.5.** *For groups of size n from $\mathcal{H}(\mathcal{A},\mathcal{E})$, there is an algorithm in time $O(n^6)$ testing isomorphism.*

The rest of the paper is organized as follows. Section 2 contains the preliminaries. In Section 3 we present the decomposition procedure into normal and complement parts , proving Theorem 1.1. In Section 4, we first present the condition that shows how testing isomorphism of the original groups relates to that of the small groups. Then we prove Theorem 1.2, elaborate on the framework, and show that how a technique from [9] allows us to reduce from $\mathcal{H}(\prod\mathcal{E},\mathcal{E})$ to $\mathcal{H}(\mathcal{A},\mathcal{E})$. Finally, in Section 5, we introduce generalized code isomorphism, the reductions (Theorem 1.3) and show how to test isomorphism of $\mathcal{H}(\mathcal{A},\mathcal{E})$. Due to the page constraints, we only give sketches of proofs for some propositions. We refer the interested readers to a full version of this article for the detailed proofs and complete algorithms.

2 Preliminaries

In this section we introduce some preliminary concepts and notations that we will be using. We refer the reader to a standard text book [18] for basic concepts in Group theory.

An abelian group is a group with group operation commutative. Given a prime p, an abelian p-group is an abelian group of order p^k, $k\in\mathbb{Z}^+$, and an elementary abelian p-group is $\mathbb{Z}_p^k$. Every abelian group can be decomposed as direct product of cyclic groups by the fundamental theorem of abelian groups.

For a group G, we say that G is the semidirect product of N by H, for $N\lhd G$ and $H\leq G$, written as $G=N\rtimes H$, if $G=NH$ and $N\cap H=\{\mathrm{id}\}$. For a given decomposition of $G=N\rtimes H$, we call N the normal subgroup of this decomposition, and H the complement subgroup. For a given $N\lhd G$, from the definition of semidirect product it can be seen that $G=N\rtimes H$ if and only if there is a set of coset representatives of G/N closed under group operation. We use C_h^N to denote the automorphism of N induced by h by conjugating action. Formally, $C_h^N: N\to N$ by $n\to hnh^{-1}$. This gives an homomorphism of $\tau: H\to \mathrm{Aut}(N)$, by sending h to C_h^N. When we write $G=N\rtimes_\tau H$, τ is the associated homomorphism from H to $\mathrm{Aut}(N)$ acting by conjugation. Conversely, given two groups N and H, and a homomorphism $\tau: H\to\mathrm{Aut}(N)$ (we will use τ_h to denote the image of h under τ), a group G can be formed as follows: elements in G are from $N\times H$, and we let $(n,h)\cdot(n',h')=(n\tau_h(n'),hh')$. This gives a construction of (outer) semidirect product $G=N\rtimes_\tau H$.[3]

▶ **Theorem 2.1.** (Schur-Zassenhaus theorem, c.f. [18]) *Let G be a finite group of order n, and m is a Hall divisor of n. If there exists $N\lhd G$, $|N|=m$, then we have $H\leq G$ such that $G=N\rtimes H$. If H and H' are two complements of N, then H and H' are conjugate.*

Representation theory of finite groups: we list basic notions and facts about representation theory of finite groups, and we refer the reader to a standard text book [20] for further details.

For a finite group G and a vector space V, a representation of G over V is a group homomorphism $\phi: G\to\mathrm{GL}(V)$. There is always a trivial representation by mapping every element in G to 1. If the underlying field of V is $\mathbb{F}$, and V is of finite dimension d, a homomorphism $\phi: G\to \mathrm{GL}(d,\mathbb{F})$ is called a representation of G over $\mathbb{F}$ of dimension d. For

[3] Note that actually $G=N'\rtimes_\tau H'$, where $N'=\{(n,1)\mid n\in N\}$ and $H'=\{(1,h)\mid h\in H\}$. τ also maps H' to $\mathrm{Aut}(N')$ naturally. As this is a simple embedding, for convenience we write $G=N\rtimes_\tau H$.

a given representation $\phi : G \to \mathrm{GL}(d, \mathbb{F})$, a subspace of V, L is an *invariant subspace*, or a *sub-representation* if $\forall g \in G$, $\phi_g(L) = L$. $\vec{0}$ and V are called trivial invariant subspaces. A representation without non-trivial invariant subspaces is called an *irreducible representation*. If ϕ and ρ are representations of a group G over spaces V and W (over a field $\mathbb{F}$), then the direct sum $\phi \oplus \rho$ is the representation of G over $V \oplus W$ defined as: $(\phi \oplus \rho)_g(u + v) := \phi_g(u) + \rho_g(v)$ for $g \in G$. A representation is *completely reducible* if it is a direct sum of irreducible representations. Maschke's theorem states that if characteristic of $\mathbb{F}$ is 0 or coprime with $|G|$, then the representation over $\mathbb{F}$ is completely reducible.

Two representations $\phi : G \to \mathrm{GL}(V)$ and $\psi : G \to \mathrm{GL}(V)$ are equivalent if there exists a general linear map $T : V \to V$ such that $\phi(g) = T\psi(g)T^{-1}$ for every $g \in G$. A fact about completely reducible representations is that two representations are equivalent if and only if irreducible representations (up to equivalence) that appear in their decompositions are the same. Specifically, decomposing a representation gives for every irreducible representation (up to equivalence) its multiplicity in that representation, and two representations are equivalent if and only if for every irreducible representation the multiplicities are the same. For a representation $\phi : G \to \mathrm{GL}(\mathbb{F}, d)$, and $i \in [d]$, let $L_\phi(i)$ be the set of irreducible representations with multiplicity i in the decomposition ϕ, and $L_\phi = (L_\phi(i))_{i\in[d]}$. We say $L_\phi = L_\psi$ if and only if $L_\phi(i) = L_\psi(i)$ for every $i \in [d]$.

We use this straightforward criterion to test whether a representation is irreducible.

▶ Proposition 1. Let $\phi : G \to \mathrm{GL}(V)$ be a representation. ϕ is irreducible if and only if $\forall v \in V$, $v \neq \vec{0}$, $\langle gv \mid g \in G\rangle = V$.

▶ **Theorem 2.2.** (Maschke's theorem. Adaptation of [20], page 6, Theorem 1) *Let* $\phi : G \to \mathrm{GL}(\mathbb{F}, d)$ *be a representation,* $\gcd(|G|, \mathrm{char}(\mathbb{F})) = 1$. $W \leq V$ *is a sub-representation of* V*. Let* $p : V \to W$ *be a projection of* V *onto* W*, and the image of* $p' = \frac{1}{|G|}\sum_{g\in G} \phi(g)\circ p\circ\phi(g^{-1})$ *be* W'*. Then* W' *is a sub-representation and* $V = W \oplus W'$*.*

Proposition 1 and Theorem 2.2 suggest the following procedure to decompose a representation into its irreducible components. Let $\phi : G \to \mathrm{GL}(V)$ be a representation. For every $v \in V$, test if $\langle gv \mid g \in G\rangle$ generates V. If so, it is an irreducible representation. Otherwise, for a specific v, $\langle gv \mid g \in G\rangle$ is a sub-representation W. Then Theorem 2.2 helps to identify a sub-representation W' such that $V = W \oplus W'$. Recursively using the above procedure on W and W' decomposes V into its irreducible components. This gives:

▶ Proposition 2. Given a representation $\phi : G \to \mathrm{GL}(V)$, its irreducible components can be listed in time $O(\dim(V)^2 \cdot |V| \cdot |G|)$.

Proposition 2 is sufficient for our purpose. But we remark that, in general, the decomposition of modular representation (representations over fields of finite characteristic) can be done much more efficient (c.f. [17] and Chapter 7.4 of [10]). Given two irreducible representations, there is an efficient algorithm to determine whether they are equivalent (c.f.[10], Chapter 7.5.3). For factoring polynomials of degree n over $\mathbb{Z}_p$, we use the $O(p^{1/2}(\log p)^2 n^{2+\varepsilon})$ algorithm in [21]. For computing canonical normal form of a linear transformation, Steel's algorithm [22] in time $O(n^4)$ suffices.

3 Decomposition into normal and complement parts

In this section we describe that for a given group, all its normal Hall subgroups and their complements can be listed, proving Theorem 1.1, by providing the following two propositions.

▶ Proposition 3. Let G be a group of size n. For a Hall divisor m, if a normal Hall subgroup of order m exists then it can be computed in time $O(n^3)$.

▶ Proposition 4. Let G be a group of order n, and N a normal Hall subgroup of order m. Then a complement of N can be found in time $O(n^4)$.

The two propositions give a natural way of listing the normal Hall subgroups and their complements: for a given Hall divisor m of the group size n, compute the normal Hall subgroup of size m by Proposition 3 if it exists. Then compute its complement by Proposition 4. Going over all Hall divisors lists all normal Hall subgroups and their complements.

Proposition 3 follows from that for a specific Hall divisor m, if the normal Hall subgroup of m exists then it is generated by $\langle g^{n/m} \mid g \in G\rangle$. Proof of Proposition 4 follows from the constructive proof of Schur-Zassenhaus theorem [18], which can be rephrased as a recursive algorithm. The base case of the algorithm is abelian groups, for which a complement can be found starting with an arbitrary set of representatives. When the input is not abelian, the algorithm branches into two cases depending on whether the normal subgroup is minimal. The case using the Hall condition is when the normal subgroup is minimal, and we use the Frattini argument and second isomorphism theorem to reduce to an instance of smaller size.

4 Condition for isomorphism testing

The next theorem shows how isomorphism of big groups reduces to that of components for groups with normal Hall subgroups. This has been discovered by Taunt [23] in the context of construction of finite groups, though he did not apply it to normal Hall subgroups explicitly.

▶ **Theorem 4.1.** *(Theorem 3.3, [23]) Given $G_1 = N_1 \rtimes_\tau H_1$, $G_2 = N_2 \rtimes_\gamma H_2$, with $|N_1| = |N_2|$, $|H_1| = |H_2|$. N_1 and N_2 are normal Hall. Then $G_1 \cong G_2$ if and only if there exist an isomorphism $\psi : N_1 \to N_2$, and an isomorphism $\phi : H_1 \to H_2$, such that, $\forall h \in H_1$,*

$$\tau(h) = \psi^{-1} \circ \gamma(\phi(h)) \circ \psi. \qquad (1)$$

4.1 Proof of Theorem 1.2

Theorem 1.2 states that isomorphism of $\mathcal{H}(\mathcal{E}, \mathcal{K})$ is equivalent to AutoInducedRepEquiv. In this section we show the two reductions here.

Isomorphism of groups in $\mathcal{H}(\mathcal{E}, \mathcal{K})$ to AutoInducedRepEquiv*:* By listing all normal Hall subgroups and their complements we can find two normal Hall subgroups of the same size from two groups. Then to test isomorphism of the original group, we first use known isomorphism procedure for normal and complement parts. Given the isomorphisms of the normal and complement parts, the only task left is to test Equation 1, which, by composing the isomorphisms of the normal and complement parts, becomes AutoInducedRepEquiv naturally.

AutoInducedRepEquiv *to isomorphism of groups in $\mathcal{H}(\mathcal{E}, \mathcal{K})$:* In Section 2 we described the standard construction that, given groups N, H and $\tau : H \to \mathrm{Aut}(N)$, defines a group $G = N \rtimes_\tau H$. Thus, given two representations τ and γ of H over $\mathbb{Z}_p^k$, we can construct $G_1 = \mathbb{Z}_p^k \rtimes_\tau H$ and $G_2 = \mathbb{Z}_p^k \rtimes_\gamma H$, and then call the oracle to test if G_1 and G_2 are isomorphic. By Theorem 4.1, the two representations are equivalent up to automorphism action if and only if G_1 and G_2 are isomorphic. This gives the reduction.

4.2 A framework for testing isomorphism of groups from $\mathcal{H}(\mathcal{K}, \mathcal{K})$

Suppose we want to test isomorphism of two groups G_1 and G_2 from $\mathcal{H}(\mathcal{K}, \mathcal{K})$. Given Theorem 1.1, for any group all its normal Hall subgroups can be listed efficiently, so we can first compare the orders of the normal Hall subgroups of G_1 and G_2, and output "not isomorphic" if there are no normal Hall subgroups of the same size. For normal Hall subgroups with the same order, compute their complements using Proposition 4. Suppose we decompose $G_1 = N_1 \rtimes H_1$ and $G_2 = N_2 \rtimes H_2$, with $|N_1| = |N_2|$. As the normal and complement parts are from groups with known isomorphism computing procedure, run the isomorphism tests between N_1, N_2 and H_1, H_2. If they are not isomorphic output "not isomorphic". Now the only task left is to test Equation 1. Recall that $\prod \mathcal{E}$ denotes the class of direct products of elementary abelian groups. The cases $\mathcal{H}(\mathcal{E}, \mathcal{B})$ and $\mathcal{H}(\prod \mathcal{E}, \mathcal{B})$ are immediate: for $\mathcal{H}(\mathcal{E}, \mathcal{B})$, the automorphisms of complements can be enumerated. For a given automorphism of the complement, the problem is to test if two representations are equivalent. It can be solved by decomposing the representations, and then noticing that equivalence of irreducible representations can be determined efficiently. For $\mathcal{H}(\prod \mathcal{E}, \mathcal{B})$, like in $\mathcal{H}(\mathcal{E}, \mathcal{B})$, as the automorphisms of the complement can be enumerated, for a given automorphism, the problem is to test if the representations over the direct factors of the normal subgroup are equivalent. These instances can be solved separately.

We remark that when the complement is in $\mathcal{B}$, to find the complement it is easy to come up with an efficient enumeration procedure (without using algorithmic Schur-Zassenhaus). It is also noted that when the normal subgroup is $\prod \mathcal{E}$, the idea of treating the representations over the factors separately does not work in general unless an automorphism of the complements is fixed as a priori. From the above discussion, the difficult case is when the complement has no generating set of size $O(1)$.

4.3 From $\mathcal{H}(\prod \mathcal{E}, \mathcal{K})$ to $\mathcal{H}(\mathcal{A}, \mathcal{K})$: Le Gall's technique

In [9], Le Gall presented a technique that reduces testing conjugation of automorphisms of an abelian group to that of linear mappings, when the orders of the automorphisms are coprime with that of the abelian group. We refer it as *Le Gall's technique* in this paper.

▶ **Lemma 4.2.** (Le Gall's technique) *For a given abelian p-group A, and a generating set $S \subseteq A$, let ϕ_1 and ϕ_2 be two automorphisms of A, given by listing the images of the generating set. If $p \nmid |\phi_1| = |\phi_2|$, there exists an efficiently-computable map $\Lambda_p : \mathrm{Aut}(A) \to \mathrm{GL}(\mathbb{Z}_p, |S|)$, such that ϕ_1 and ϕ_2 are conjugate if and only if $\Lambda_p(\phi_1)$ and $\Lambda_p(\phi_2)$ are conjugate.*

We show that Le Gall's technique allows us to reduce testing isomorphism of $\mathcal{H}(\mathcal{A}, \mathcal{K})$ to that of $\mathcal{H}(\prod \mathcal{E}, \mathcal{K})$. For convenience we first explain how Le Gall's technique allows us to reduce isomorphism of $\mathcal{H}(\mathcal{A}_p, \mathcal{K})$ to $\mathcal{H}(\mathcal{E}, \mathcal{K})$. Let G_1 and G_2 be decomposed as $N_1 \rtimes_\tau H_1$ and $N_2 \rtimes_\gamma H_2$, where N_1 and N_2 are abelian p-groups. Then decompose N_1 and N_2 into the canonical form, and identify H_1 and H_2 as isomorphic. Now by Theorem 4.1, we need to test if there exist $\psi \in \mathrm{Aut}(N_1)$, and $\phi \in \mathrm{Aut}(H)$, such that $\tau(h)$ and $\gamma(\phi(h))$ are conjugate by ψ, for every $h \in H$. Noting that $p \nmid |H|$, Lemma 4.2 tells that this happens if and only if $\Lambda_p(\tau(h))$ and $\Lambda_p(\gamma(\phi(h)))$ are conjugate. Thus composing Λ_p with τ and γ, noting that $\Lambda_p \circ \tau$ and $\Lambda_p \circ \gamma$ send H to $\mathrm{GL}(\mathbb{Z}_p, k)$, we reduce the case of $\mathcal{H}(\mathcal{A}_p, \mathcal{K})$ to $\mathcal{H}(\mathbb{Z}_p^k, \mathcal{K})$. To go from $\mathcal{H}(\mathcal{A}, \mathcal{K})$ to $\mathcal{H}(\prod \mathcal{E}, \mathcal{K})$ we just need to consider the factors of $\prod \mathcal{E}$ separately and apply the appropriate Λ_p.

5 Isomorphism of $\mathcal{H}(\mathcal{A}, \mathcal{E})$

The main result of this section is a reduction of the isomorphism testing problem for groups in $\mathcal{H}(\mathcal{A}, \mathcal{E})$ to the problem of generalized code isomorphism problem. We first introduce this problem. For $\mathbb{F}^n$, a linear code of dimension d is a subspace of dimension d. A generating matrix of a code C of dimension d is a d by n matrix with row vectors being a basis of

C. With abuse of notation we will also use C to denote the generating matrix of the code C. Two codes C and D of dimension d over $\mathbb{F}$ are isomorphic if they are equivalent up to permutation of coordinates. Formally, if there exists a d by d non-singular matrix G and an n by n permutation matrix P such that $GCP = D$.

▶ **Theorem 5.1.** *([3]) For C and D be two linear codes given as generating matrices, their isomorphism can be tested, and the coset of isomorphism be computed, in time $(2 + o(1))^n$.*

We generalizes code isomorphism problem slightly to get:

▶ Problem 2. *(Generalized code isomorphism problem)* Given two matrices $d' \times n$ matrices C' and D' over the field $\mathbb{F}$, and a permutation group $S \leq S_n$, if there exists $G \in \mathrm{GL}(\mathbb{F}, d')$ and a permutation matrix $P \in S$, such that $GC'P = D'$.

The generalized code isomorphism problem generalizes code isomorphism problem in two ways: first we do not require row vectors of C' and D' to be linearly independent. Secondly the permutation matrix P must come from a certain permutation group S. Its solution in singly exponential time can be viewed as a corollary to Theorem 5.1, by applying a coset intersection running in singly exponential time[2].

▶ **Corollary 5.2.** *Given two $d' \times n$ matrices C' and D', and a permutation group S, whether C' and D' are isomorphic can be tested, the coset of permutation matrices be computed, in time $(2 + o(1))^n$.*

5.1 Representation of $\mathbb{Z}_q^\ell$ over $\mathbb{Z}_p$

In this section, we recall basic facts concerning representations of $\mathbb{Z}_q^\ell$ over $\mathbb{Z}_p$, p, q two different primes, and we refer the reader to standard textbooks for more details. First suppose the cyclotomic polynomial $\Phi_q(x)$ factors as $g_1 \cdot g_2 \cdot \ldots \cdot g_r$ over $\mathbb{Z}_p$, in which g_i's are monic polynomials with the same degree $d = (q-1)/r$. It is noted that d is the order of p in the multiplicative group $(\mathbb{Z}/q\mathbb{Z})^\times$. Let $M \in \mathrm{GL}(\mathbb{Z}_p, d)$ be the companion matrix of g_1.[4] For $v \in \mathbb{Z}_q^\ell$, $v \neq \vec{0}$, we define $v^* : \mathbb{Z}_q^\ell \to \mathbb{Z}_q$ by mapping $v^*(u) = (v, u)$ (the inner product of v and u). Now define $f_v : \mathbb{Z}_q^\ell \to \mathrm{GL}(\mathbb{Z}_p, d)$ by sending $u \to M^{v^*(u)}$. To unify notation let $f_{\vec{0}} : \mathbb{Z}_q^\ell \to \mathbb{Z}_p$ be the trivial representation. Then f_v gives an irreducible representation of $\mathbb{Z}_q^\ell$ over $\mathbb{Z}_p$, and $\{f_v \mid v \in V\}$ is the set of all irreducible representations. However, f_v and f_u may be equivalent, for $u, v \in V$, as described in the following claim.

▶ Claim 1. Let f_v and f_u be two irreducible representations of $\mathbb{Z}_q^\ell$ over $\mathbb{Z}_p$ induced from $v, u \in \mathbb{Z}_q^\ell$, $v, u \neq \vec{0}$ as above. f_v and f_u are equivalent if and only if $u = sv$ for $s \in \mathbb{Z}_q$, and M^s and M are conjugate.

▶ **Corollary 5.3.** *Let $S_{p,q}$ be the set of s satisfying the condition in Claim 1, and d be the order of p in the multiplicative group $(\mathbb{Z}/q\mathbb{Z})^\times$. Then $|S_{p,q}| = d$.*

Let $\tau : \mathbb{Z}_q^\ell \to \mathrm{GL}(\mathbb{Z}_p, k)$ be a representation. Due to Maschke's theorem, representations of $\mathbb{Z}_q^\ell$ over $\mathbb{Z}_p$ are completely reducible. Suppose $\tau = f_{v_1}^{k_1} \oplus \cdots \oplus f_{v_t}^{k_t}$, for $v_i \in V$, $i \in [t]$, $k_1 \geq \cdots \geq k_t \geq 1$. Note that t is bounded by $1 + \lfloor (k-1)/d \rfloor$ or k/d, depending on whether the trivial representation exists or not. We will assume when a representation is decomposed as such, the multiplicities of irreducible components are arranged to be non-increasing. For

[4] In fact, any d by d matrix with characteristic polynomial as g_1 would suffice, and it does not matter if we choose, say companion matrix of g_i, for any $i \in [r]$.

a given multiplicity $w \in [k]$, recall that $L_\tau(w)$ is the set of irreducible representations with multiplicity w appearing in τ, and $L_\tau = (L_\tau(w))_{w\in[k]}$ determines a representation up to equivalence. The problem of working with L_τ is that the irreducible representations are "abstract", while we need to actually know the form of the irreducible representations. The idea is to use vectors to index irreducible representations, at the cost of losing uniqueness.

▶ **Definition 5.4.** Given a representation $\tau : \mathbb{Z}_q^\ell \to \mathrm{GL}(\mathbb{Z}_p, k)$, and $w \in [k]$, $\mathcal{L}_\tau(w)$ is a set of vectors such that for every irreducible representation $f \in L_\tau(w)$, there is a unique vector $v \in \mathcal{L}_\tau(w)$ such that f_v and f are equivalent. $\mathcal{L}_\tau = (\mathcal{L}_\tau(w))_{w\in[k]}$. Such a tuple of sets of vectors is called an *indexing tuple* of L_τ.

▶ Remark. By Corollary 5.3, the number of different indexing tuples of L_τ is bounded by $d^{k/d} \leq (e^{1/e})^k < 2^k$. (Note that we do not need to consider $f_{\vec{0}}$.)

For two representations $\tau : \mathbb{Z}_q^\ell \to \mathrm{GL}(\mathbb{Z}_p, k)$ and $\gamma : \mathbb{Z}_q^\ell \to \mathrm{GL}(\mathbb{Z}_p, k)$, τ and γ are equivalent if and only if $L_\tau = L_\gamma$. For two indexing tuples $\mathcal{L}_\tau$ and $\mathcal{L}_\gamma$ of τ and γ, we also use $\mathcal{L}_\tau = \mathcal{L}_\gamma$ to denote for every $w \in [k]$, $\mathcal{L}_\tau(w) = \mathcal{L}_\gamma(w)$. An immediate consequence is the following claim.

▶ Claim 2. Let $\tau : \mathbb{Z}_q^\ell \to \mathrm{GL}(\mathbb{Z}_p, k)$ and $\gamma : \mathbb{Z}_q^\ell \to \mathrm{GL}(\mathbb{Z}_p, k)$ be two representations. τ and γ are equivalent if and only if there exist indexing tuples of τ and γ, $\mathcal{L}_\tau$ and $\mathcal{L}_\gamma$, such that $\mathcal{L}_\tau = \mathcal{L}_\gamma$.

The induced representation of f_v by $\phi \in \mathrm{GL}(\mathbb{Z}_q, l)$ has a nice form: $(f_v \circ \phi)(u) = f_v(\phi(u)) = M^{v^*(\phi(u))} = M^{(\phi^T(v))^*(u)} = f_{\phi^T(v)}(u)$. That is $f_v \circ \phi = f_{\phi^T(v)}$. Note that for any two representations g and h of an arbitrary group G and $\phi' \in \mathrm{Aut}(G)$, $(g \oplus h) \circ \phi' = (g \circ \phi') \oplus (h \circ \phi')$. If follows that $\tau \circ \phi = f_{\phi^T(v_1)}^{k_1} \oplus \cdots \oplus f_{\phi^T(v_t)}^{k_t}$. For $\phi \in \mathrm{GL}(\mathbb{Z}_q, l)$, and $S \subseteq \mathbb{Z}_q^\ell$, S^ϕ is the set obtained by applying ϕ^T to every vector in S. Thus $\mathcal{L}_{\tau\circ\phi} = \mathcal{L}_\tau^\phi \doteq (\mathcal{L}_\tau(w)^\phi \mid w \in [k])$.

5.2 Isomorphism of $\mathcal{H}(\mathcal{E}, \mathcal{E})$: proof of Theorem 1.3

To test isomorphism of two groups G_1 and G_2 identified as $\mathbb{Z}_p^k \rtimes_\tau \mathbb{Z}_q^\ell$ and $\mathbb{Z}_p^k \rtimes_\gamma \mathbb{Z}_q^\ell$, by Theorem 1.2 we can view τ and γ as two representations of $\mathbb{Z}_q^\ell$ over $\mathbb{Z}_p$ of dimension k. Then we need to solve AutoInducedRepEquiv problem for τ and γ. This is done, as shown in Theorem 1.3, by reducing to generalized code isomorphism problem.

Since τ and γ are equivalent if and only if $L_\tau = L_\gamma$, using Proposition 2 we decompose τ and γ as $\tau = f_{v_1}^{k_1} \oplus \cdots \oplus f_{v_t}^{k_t}$ and $\gamma = f_{u_1}^{\ell_1} \oplus \cdots \oplus f_{u_{t'}}^{\ell_{t'}}$ to get two specific indexing sets $\mathcal{L}_\tau$ and $\mathcal{L}_\gamma$. Along with the decomposition, we can calculate the change of basis matrices S and T, such that, the images of $S(\tau \circ \phi)S^{-1}$ and $T\gamma T^{-1}$ are sets of block diagonal matrices with blocks representing the irreducible representations. Also note that for a specific irreducible representation, it is easy to identify an indexing vector of it, by examining which vector maps to M, the companion matrix of some pre-determined factor of $\Phi_q(x)$ over $\mathbb{Z}_p$.

Given the decomposition, we first need to test if $t = t'$, and $|\mathcal{L}_\tau(w)| = |\mathcal{L}_\gamma(w)|$, $\forall w \in [k]$. If the conditions are not satisfied τ and γ can not be equivalent under automorphism. For now assume that the conditions are satisfied. By $\mathcal{L}_{\tau\circ\phi} = \mathcal{L}_\tau^\phi$, we know the indexing tuple of $\mathcal{L}_{\tau\circ\phi}$ is to apply ϕ^T to the vectors in $\mathcal{L}_\tau$. From a specific indexing tuple $\mathcal{L}_\tau$, all indexing tuples of L_τ can be enumerated based on Claim 1. From Remark 5.1, we can afford the enumeration of all indexing tuples. Finally, by Claim 2, the only task left is to determine whether there exists $\phi \in \mathrm{GL}(\mathbb{Z}_p, \ell)$, such that $\mathcal{L}_\tau^\phi$ is a specific indexing tuple of L_γ, in time $\mathrm{poly}(p^k, q^\ell)$, where $p^k \cdot q^\ell$ is the size of the original group.

▶ Proposition 5. Testing the existence of ϕ so that of $\mathcal{L}_\tau^{\phi^T} = \mathcal{L}_\gamma$ in time $\mathrm{poly}(p^k, q^\ell)$ reduces to generalized code isomorphism problem in singly exponential time.

Proof. Expand $\mathcal{L}_\tau = (\mathcal{L}_\tau(1), \ldots, \mathcal{L}_\tau(k))$ as

$$(\{v_1, \ldots, v_{s_1}\}, \{v_{s_1+1}, \ldots, v_{s_2}\}, \ldots, \{v_{s_{k-1}+1}, \ldots, v_{s_k}\}),$$

in which $s_1 \leq s_2 \leq \cdots \leq s_k = t$. Similarly expand $\mathcal{L}_\gamma$ as

$$(\{u_1, \ldots, u_{s_1}\}, \{u_{s_1+1}, \ldots, u_{s_2}\}, \ldots, \{u_{s_{k-1}+1}, \ldots, u_{s_k}\}).$$

$\mathcal{L}_\tau^{\phi^T}$ is just $(\{\phi(v_1), \ldots, \phi(v_{s_1})\}, \{\phi(v_{s_1+1}), \ldots, \phi(v_{s_2})\}, \ldots, \{\phi(v_{s_{k-1}+1}), \ldots, \phi(v_{s_k})\})$, $\mathcal{L}_\tau^{\phi^T} = \mathcal{L}_\gamma$ can be formulated as finding $\phi \in \mathrm{GL}(\mathbb{Z}_q, \ell)$ and $\sigma \in S_{s_1} \times S_{s_2-s_1} \times \cdots \times S_{s_k - s_{k-1}}$ such that $\phi(v_1, \ldots, v_t)\sigma = (u_1, \ldots, u_t)$. This is just generalized code isomorphism problem with the permutation group $S_{s_1} \times S_{s_2-s_1} \times \cdots \times S_{s_k-s_{k-1}}$, whose the generators can be computed as symmetric groups can be generated by two elements. The reduction takes time poly(k, ℓ). ◀

Thus the solution for generalized code isomorphism in singly exponential time gives the algorithm for AutoInducedRepEquiv for elementary abelian groups, finishing the proof of Theorem 1.3.

5.3 Isomorphism of $\mathcal{H}(\mathcal{A}, \mathcal{E})$

The idea for $\mathcal{H}(\mathcal{E}, \mathcal{E})$ can be extended to $\mathcal{H}(\prod \mathcal{E}, \mathcal{E})$, as follows. Suppose we have G_1 and G_2 identified as $(\prod_{i\in[s]} \mathbb{Z}_{p_i}^{k_i}) \rtimes \mathbb{Z}_q^\ell$, with the associated actions as τ and γ, respectively. Now we need to test if there exist $\psi \in \prod_{i\in[s]} \mathrm{GL}(\mathbb{Z}_{p_i}, k_i)$ and $\phi \in \mathrm{GL}(\mathbb{Z}_q, l)$ such that $\tau(h) = \psi^{-1} \circ \gamma(\phi(h)) \circ \psi$, for every $h \in \mathbb{Z}_q^\ell$. Let $\tau_i : H_1 \to \mathrm{GL}(\mathbb{Z}_{p_i}, k_i)$ be the projection of τ into the ith component, and similarly we have $\gamma_i : H_2 \to \mathrm{GL}(\mathbb{Z}_{p_i}, k_i)$. This reduces to testing for every $i \in [s]$, if $\tau_i(h)$ and $\gamma_i(\phi(h))$ are conjugate by $\psi_i \in \mathrm{GL}(\mathbb{Z}_{p_i}, k_i)$, for every $h \in \mathbb{Z}_q^\ell$. Viewing τ_i's and γ_i's as representations and going through the decomposition into irreducibles, we get $\mathcal{L}_{\tau_i}$'s and $\mathcal{L}_{\gamma_j}$'s and similarly we need to determine if there exists $\phi \in \mathrm{GL}(\mathbb{Z}_q, l)$ such that $\mathcal{L}_{\tau_i}^{\phi^T} = \mathcal{L}_{\gamma_i}$, for every $i \in [s]$. Now it is enough to group $\mathcal{L}_{\tau_i}$'s and $\mathcal{L}_{\gamma_j}$'s respectively, and view them as a single generalized code isomorphism instance. Finally, Le Gall's technique gives an efficient algorithm for groups from $\mathcal{H}(\mathcal{A}, \mathcal{E})$.

Acknowledgement

Part of the work was done while Youming Qiao was visiting the University of Chicago, and he would like to thank Laci Babai and Sasha Razborov for hosting him. Youming would also like to thank J.L. Alperin and James B. Wilson for several useful discussions. The authors are also grateful to Laci Babai for sharing the results in [3].

References

1 S.I. Adian. The unsolvability of certain algorithmic problems in the theory of groups. *Trudy Moskov. Math. Obshch*, 6:231–298, 1957.

2 László Babai. Coset intersection in moderately exponential time. *Chicago J. Theoret. Comp. Sci.*, 2010.

3 Lászlo Babai. Equivalence of linear codes. Manuscript, 2010. See [4].

4 László Babai, Paolo Codenotti, Joshua Grochow, and Youming Qiao. Towards efficient algorithm for semisimple group isomorphism. Manuscript, 2010. To appear at SODA 11.

5 László Babai and Eugene M. Luks. Canonical labeling of graphs. In *STOC '83: Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 171–183, New York, NY, USA, 1983. ACM.

6 László Babai and Endre Szemerédi. On the complexity of matrix group problems i. In *FOCS*, pages 229–240, 1984.

7 Arkadev Chattopadhyay, Jacobo Toran, and Fabian Wagner. Graph isomorphism is not AC^0 reducible to Group Isomorphism. In *Proceedings of FSTTCS 2010 (To Appear)*, July 2010. Technical report available at ECCC : TR10-117.

8 Max Dehn. über unendliche diskontinuierliche gruppen. *Mathematische Annalen*, 71:116–144, 1911.

9 Francois Le Gall. Efficient isomorphism testing for a class of group extensions. In Susanne Albers and Jean-Yves Marion, editors, *26th International Symposium on Theoretical Aspects of Computer Science (STACS 2009)*, volume 3 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 625–636, Dagstuhl, Germany, 2009. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

10 Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien. *Handbook of computational group theory.* Chapman and Hall/CRC, London, 2005.

11 T. Kavitha. Linear time algorithms for abelian group isomorphism and related problems. *J. Comput. Syst. Sci.*, 73(6):986–996, 2007.

12 Neeraj Kayal and Timur Nezhmetdinov. Factoring groups efficiently. In *Proceedings of the 36th ICALP (2009)*, pages 585–596, 2009. Also availabe as ECCC Tech Report TR08-074.

13 J. Köbler, U. Schöning, and J. Torán. *The Graph Isomorphism Problem: Its Structural Complexity.* Progress in Theoretical Computer Science. Birkhauser, Boston, 1993.

14 Richard J. Lipton, Lawrence Snyder, and Y. Zalcstein. The complexity of word and isomorphism problems for finite groups. Technical report, John Hopkins, 1976.

15 Federico Menegazzo. The number of generators of a finite group. *Irish Math. Soc. Bulletin*, 50:117–128, 2003.

16 Gary L. Miller. On the $n \log n$ isomorphism technique (a preliminary report). In *STOC '78: Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 51–58, New York, NY, USA, 1978. ACM.

17 Lajos Rónyai. Computing the structure of finite algebras. *J. Symb. Comput.*, 9(3):355–373, 1990.

18 Joseph J. Rotman. *An Introduction to the Theory of Groups (4th Ed.).* Springer-Verlag, 1995.

19 C. Savage. An $O(n^2)$ algorithm for abelian group isomorphism. Technical report, North Carolina State University, 1980.

20 Jean Pierre Serre. *Linear representations of finite groups.* Springer-Verlag, New York, 1977. Translated from the second French edition by Leonard L. Scott, Graduate Texts in Mathematics, Vol. 42.

21 Victor Shoup. On the deterministic complexity of factoring polynomials over finite fields. *Inform. Process. Lett*, 33:261–267, 1990.

22 Allan Steel. A new algorithm for the computation of canonical forms of matrices over fields. *Journal of Symbolic Computation*, 24(3-4):409 – 432, 1997.

23 D. R. Taunt. Remarks on the isomorphism problem in theories of construction of finite groups. *Mathematical Proceedings of the Cambridge Philosophical Society*, 51:16–24, 1955.

24 Narayan Vikas. An $O(n)$ algorithm for abelian p-group isomorphism and an $O(n \log n)$ algorithm for abelian group isomorphism. *J. Comput. Syst. Sci.*, 53(1):1–9, 1996.

25 James B. Wilson. Decomposing p-groups via jordan algebras. *J. Algebra*, 322:2642–2679, 2009.

26 James B. Wilson. Finding central decompositions of p-groups. *J. Group Theory*, 12:813–830, 2009.

27 James B. Wilson. Finding direct product decompositions in polynomial time, May 2010. arXiv:1005.0548.

Space Complexity of Perfect Matching in Bounded Genus Bipartite Graphs

Samir Datta[1], Raghav Kulkarni[2], Raghunath Tewari[3], and N. Variyam Vinodchandran[4]

1 **Chennai Mathematical Institute**
Chennai, India
`sdatta@cmi.ac.in`

2 **University of Chicago**
Chicago, USA
`raghav@cs.uchicago.edu`

3 **University of Nebraska-Lincoln**
Lincoln, USA
`rtewari@cse.unl.edu`

4 **University of Nebraska-Lincoln**
Lincoln, USA
`vinod@cse.unl.edu`

Abstract

We investigate the space complexity of certain perfect matching problems over bipartite graphs embedded on surfaces of constant genus (orientable or non-orientable). We show that the problems of deciding whether such graphs have (1) a perfect matching or not and (2) a unique perfect matching or not, are in the logspace complexity class SPL. Since SPL is contained in the logspace counting classes $\oplus$L (in fact in $\mathrm{Mod}_k\mathrm{L}$ for all $k \geq 2$), $\mathrm{C}_=\mathrm{L}$, and PL, our upper bound places the above-mentioned matching problems in these counting classes as well. We also show that the search version, computing a perfect matching, for this class of graphs is in $\mathsf{FL}^{\mathsf{SPL}}$. Our results extend the same upper bounds for these problems over bipartite planar graphs known earlier.

As our main technical result, we design a logspace computable and polynomially bounded weight function which isolates a minimum weight perfect matching in bipartite graphs embedded on surfaces of constant genus. We use results from algebraic topology for proving the correctness of the weight function.

1998 ACM Subject Classification Computational Complexity

Keywords and phrases perfect matching, bounded genus graphs, isolation problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.579

1 Introduction

The *perfect matching* problem and its variations are one of the most well-studied problems in theoretical computer science. Research in understanding the inherent complexity of computational problems related to matching has lead to important results and techniques in complexity theory and elsewhere in theoretical computer science. However, even after decades of research, the exact complexity of many problems related to matching is not yet completely understood.

We investigate the *space complexity* of certain well studied perfect matching problems over bipartite graphs. We prove new uniform space complexity upper bounds on these problems for *graphs embedded on surfaces of constant genus.* We prove our upper bounds

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 579–590
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

by solving the technical problem of 'deterministically isolating' a perfect matching for this class of graphs.

Distinguishing a single solution out of a set of solutions is a basic algorithmic problem with many applications. The *Isolation Lemma* due to Mulmulay, Vazirani, and Vazirani provides a general randomized solution to this problem. Let $\mathcal{F}$ be a non-empty set system on $U = \{1, \ldots, n\}$. The Isolation Lemma says, for a random weight function on U (bounded by $n^{O(1)}$), with high probability there is a *unique* set in $\mathcal{F}$ of minimum weight [14]. This lemma was originally used to give an elegant RNC algorithm for constructing a maximum matching (by isolating a minimum weight perfect matching) in general graphs. Since its discovery, the Isolation Lemma has found many applications, mostly in discovering new randomized or non-uniform upper bounds, via isolating minimum weight solutions [14, 15, 8, 1]. Clearly, derandomizing the Isolation Lemma in sufficient generality will improve these upper bounds to their deterministic counterparts and hence will be a major result. Unfortunately, recently it is shown that such a derandomization will imply certain circuit lower bounds and hence is a difficult task [3].

Can we bypass the Isolation Lemma altogether and deterministically isolate minimum weight solutions in specific situations? Recent results illustrate that one may be able to use the structure of specific computational problems under consideration to achieve non-trivial deterministic isolation. In [4], the authors used the structure of directed paths in planar graphs to prescribe a simple weight function that is computable deterministically in logarithmic space with respect to which the minimum weight directed path between any two vertices is unique. In [6], the authors isolated a perfect matching in planar bipartite graphs. In this paper we extend the deterministic isolation technique of [6] to isolate a minimum weight perfect matching in bipartite graphs embedded on constant genus surfaces.

Our Contribution

Let G be a bipartite graph with a weight function w on it edges. For an even cycle $C = e_1 e_2 \cdots e_{2k}$, the circulation of C with respect to w is the sum $\sum_{i=1}^{2k}(-1)^i w(e_i)$. The main technical contribution of the present paper can be stated (semi-formally) as follows.

Main Technical Result. There is a logspace matching preserving reduction f, and a logspace computable and polynomially bounded weight function w, so that given a bipartite graph G with a combinatorial embedding on a surface of constant genus, the circulation of any simple cycle in $f(G)$ with respect to w is non-zero. (This implies that the minimum weight perfect matching in $f(G)$ is unique [6]).

We use this result to establish (using known techniques) the following new upper bounds. Refer to the next section for definitions.

New Upper Bounds. For bipartite graphs, combinatorially embedded on surfaces of constant genus the problems Decision-BPM and Unique-BPM are in SPL, and the problem Search-BPM is in $\mathsf{FL}^{\mathsf{SPL}}$.

SPL is a logspace complexity class that was first studied by Allender, Reinhardt, and Zhou [1]. This is the class of problems reducible to the determinant with the promise that the determinant is either 0 or 1. In [1], the authors show, using a non-uniform version of Isolation Lemma, that perfect matching problem for general graphs is in a 'non-uniform' version of SPL. In [6], using the above-mentioned deterministic isolation, the authors show that for planar bipartite graphs, Decision-BPM is in fact in SPL (uniformly). Recently, Hoang showed that for graphs with polynomially many matchings, perfect matchings and many related matching problems are in SPL [9]. SPL is contained in logspace counting classes such as $\mathrm{Mod}_k\mathrm{L}$ for all $k \geq 2$ (in particular in $\oplus\mathrm{L}$), PL, and $\mathrm{C}_=\mathrm{L}$, which are in turn contained in NC^2. Thus the upper bound of SPL that we prove implies that the problems Decision-BPM and Unique-BPM for the class of graphs we study are in these logspace counting classes as well.

The techniques that we use in this paper can also be used to isolate directed paths in graphs on constant genus surfaces. This shows that the reachability problem for this class of graphs can be decided in the unambiguous class UL, extending the results of [4]. But this upper bound is already known since recently Kynčl and Vyskočil show that reachability for bounded genus graphs logspace reduces to reachability in planar graphs [11].

Matching problems over graphs of low genus have been of interest to researchers, mainly from a parallel complexity viewpoint. The matching problems that we consider in this paper are known to be in NC. In particular in [10], the authors present an NC^2 algorithm for computing a perfect matching for bipartite graphs on surfaces of $O(\log n)$ genus (readers can also find an account of known parallel complexity upper bounds for matching problems over various classes of graphs in their paper). However, the space complexity of matching problems for graphs of low genus has not been investigated before. The present paper takes a step in this direction.

Proof Outline. We assume that the graph G is presented as a combinatorial embedding on a surface (orientable or non-orientable) of genus g, where g is a constant. This is a standard assumption when dealing with graphs on surfaces, since it is NP-complete to check whether a graph has genus $\leq g$ [16]. We first give a sequence of two reductions to get, from G, a graph G' with an embedding on a genus g 'polygonal schema in normal form'. These two reductions work for both orientable and non-orientable cases. At this point we take care of the non-orientable case by reducing it to the orientable case. Once we have the embedding on an orientable polygonal schema in normal form, we further reduce G' to G'' where G'' is embedded on a constant genus 'grid graph'. These reductions are matching preserving, bipartiteness preserving and computable in logspace. Finally, for G'', we prescribe a set of $4g+1$ weight functions, $\mathcal{W} = \{w_i\}_{1\leq i\leq 4g+1}$, so that for any cycle C in G'', there is a weight function $w_i \in \mathcal{W}$ with respect to which the circulation of C is non-zero. Since g is constant, we can take a linear combination of the elements in $\mathcal{W}$, for example $\sum_{w_i\in\mathcal{W}} w_i \times (n^c)^i$ (where n is the number of vertices in the grid) for some fixed constant c (say $c = 4$), to get a single weight function with respect which the circulation of any cycle is non-zero.

The intuition behind these weight functions is as follows (for some of the definitions, refer to later sections). The set $\mathcal{W}$ is a disjoint union $\mathcal{W}_1 \cup \mathcal{W}_2 \cup \{w\}$ of the sets of weight functions $\mathcal{W}_1$, $\mathcal{W}_2$, and $\{w\}$. Consider a graph G embedded on a fundamental polygon with $2g$ sides. There are two types cycles in G: *surface separating* and *surface non-separating.* A basic theorem from algebraic topology implies that a surface non-separating cycle will intersect at least one of the sides of the polygon an odd number of times. This leads to $2g$ weight functions in $\mathcal{W}_1$ to take care of all the surface non-separating cycles. There are two types of surface separating cycles: (a) ones which completely lie inside the polygon and (b) the ones which cross some boundary. Cycles of type (a) behave exactly like cycles in the plane so the weight function w designed for planar graphs works (from [6]). For dealing with cycles of type (b), we first prove that if such a cycle intersects a boundary, it should alternate between 'coming in' and 'going out'. This leads to $2g$ weight functions in $\mathcal{W}_2$ which handle all type (b) cycles.

Figure 1 gives a pictorial view of the components involved in the proof of our main technical result.

The rest of the paper is organized as follows. In Section 2 we give the necessary definitions and state results from earlier work, that we use in this paper. In Section 3 we give matching preserving, logspace reductions from a combinatorial embedding of the graph on a surface of genus g, to a grid embedding. Due to space constraints we omit the proof of the reductions (for more details regarding the proofs please refer to the ECCC version of this paper [7]). In Section 4 we state and prove our upper bounds assuming a grid embedding. In Section 5 we reduce the non-orientable case to the orientable one.

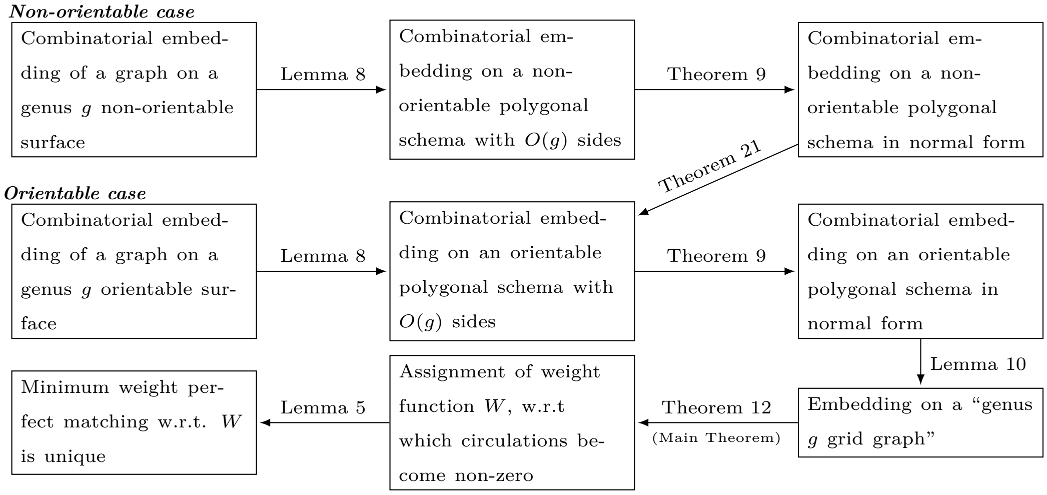

Non-orientable case
Combinatorial embedding of a graph on a genus g non-orientable surface
Lemma 8
Combinatorial embedding on a non-orientable polygonal schema with $O(g)$ sides
Theorem 9
Combinatorial embedding on a non-orientable polygonal schema in normal form
Theorem 21
Orientable case
Combinatorial embedding of a graph on a genus g orientable surface
Lemma 8
Combinatorial embedding on an orientable polygonal schema with $O(g)$ sides
Theorem 9
Combinatorial embedding on an orientable polygonal schema in normal form
Lemma 10
Embedding on a "genus g grid graph"
Theorem 12
(Main Theorem)
Assignment of weight function W, w.r.t which circulations become non-zero
Lemma 5
Minimum weight perfect matching w.r.t. W is unique

Figure 1 Outline of the steps. Note that all reductions are matching preserving and logspace computable.

2 Preliminaries

2.1 Topological graph theory

We introduce the necessary terminology from algebraic topology. For a more comprehensive understanding of this topic, refer to any standard algebraic topology book such as [12].

A *2-manifold* is a topological space such that every point has an open neighborhood homeomorphic to $\mathbb{R}^2$ and two distinct points have disjoint neighborhoods. A 2-manifold is often called a *surface*. The *genus* of a surface Γ is the maximum number g, such that there are g cycles $C_1, C_2, \ldots, C_g$ on Γ, with $C_i \cap C_j = \emptyset$ for all i, j and $\Gamma \setminus (C_1 \cup C_2 \cup \ldots \cup C_g)$ is connected. A surface is called *orientable* if it has two distinct sides, else it is called *non-orientable*. A cycle C in Γ is said to be *non-separating* if there exists a path between any two points in $\Gamma \setminus C$, else it is called *separating*.

A *polygonal schema* of a surface Γ, is a polygon with $2g'$ directed sides, such that the sides of the polygon are partitioned into g' classes, each class containing exactly two sides and glueing the two sides of each equivalence class gives the surface Γ (upto homeomorphism). A side in the ith equivalence class is labelled σ_i or $\bar{\sigma}_i$ depending on whether it is directed clockwise or anti-clockwise respectively. The *partner* of a side σ is the other side in its equivalence class. By an abuse of notation, we shall sometimes refer to the symbol of a side's partner, as the partner of the symbol. Frequently we will denote a polygonal schema as a linear ordering of its sides moving in a clockwise direction, denoted by X. For a polygonal schema X, we shall refer to any polygonal schema which is a cyclic permutation, or a reversal of the symbols, or a complementation (σ mapped to $\bar{\sigma}$ and vice versa) of the symbols, as being the same as X. A polygonal schema is called orientable (resp. non-orientable) if the corresponding surface is orientable (resp. non-orientable).

▶ **Definition 1.** An orientable polygonal schema is said to be in *normal form* if it is in one of the following forms:

$$\sigma_1\tau_1\bar{\sigma_1}\bar{\tau_1}\sigma_2\tau_2\bar{\sigma_2}\bar{\tau_2}\dots\sigma_m\tau_m\bar{\sigma_m}\bar{\tau_m} \tag{2.1}$$
$$\sigma\bar{\sigma} \tag{2.2}$$

A non-orientable polygonal schema is said to be in normal form if it is of one of the following forms:

$$\sigma\sigma X \tag{2.3}$$
$$\sigma\tau\bar{\sigma}\tau X \tag{2.4}$$

where, X is a string representing an orientable schema in normal form (i.e. like Form 2.1 or 2.2 above) or possibly an empty string.

We denote the polygonal schema in the normal form of a surface Γ as $\Lambda(\Gamma)$. We will refer to two orientable symbols σ, τ which form the following contiguous substring: $\sigma\tau\bar{\sigma}\bar{\tau}$ as being clustered together while a non-orientable symbol σ which occurs like $\sigma\sigma$ as a contiguous subtring is said to form a *pair*. Thus, in the first and third normal forms above all symbols are clustered. The first normal form represents a connected sum of torii and the third of a projective plane and torii. In the fourth normal form all but one of the orientable symbols are clustered while the only non-orientable symbol is sort of clustered with the other orientable symbol. This form represents a connected sum of a Klein Bottle and torii. The second normal form represents a sphere.

We next introduce the concept of $\mathbb{Z}_2$-homology. Given a 2-manifold Γ, a *1-cycle* is a closed curve in Γ. The set of 1-cycles forms an Abelian group, denoted as $\mathcal{C}_1(\Gamma)$, under the *symmetric difference* operation, Δ. Two 1-cycles C_1, C_2 are said to be homologically equivalent if $C_1 \Delta C_2$ forms the boundary of some region in Γ. Observe that this is an equivalence relation. Then the *first homology group* of Γ, $H_1(\Gamma)$, is the set of equivalence classes of 1-cycles. In other words, if $\mathcal{B}_1(\Gamma)$ is defined to be the subset of $\mathcal{C}_1(\Gamma)$ that are homologically equivalent to the empty set, then $H_1(\Gamma) = \mathcal{C}_1(\Gamma)/\mathcal{B}_1(\Gamma)$. If Γ is a genus g surface then $H_1(\Gamma)$ is generated by a system of $2g$ 1-cycles, having only one point in common, and whose complement is homeomorphic to a topological disk. Such a disk is also referred to as the *fundamental polygon* of Γ.

An undirected graph G is said to be embedded on a surface Γ if it can be drawn on Γ so that no two edges cross. We assume that the graph is given with a *combinatorial embedding* on a surface of constant genus. Refer to the book by Mohar and Thomassen [13] for details. The genus of a graph G is the minimum number g such that G has an embedding on a surface of genus g. We shall also refer to such an embedding as the *minimal embedding* of G. A genus g graph is said to be orientable (non-orientable) if the surface is orientable (non-orientable). A *2-cell embedding* of a graph is a combinatorial embedding of the graph on a surface such that every face is homeomorphic to the disk. Note that a minimal embedding of a graph is always a 2-cell embedding but the converse is not true. For our purposes it is enough to assume a 2-cell embedding of the given graph.

▶ **Definition 2.** The *polygonal schema of a graph* G is a combinatorial embedding given on the polygonal schema of some surface Γ together with the ordered set of vertices on each side of the polygon. Formally it is a tuple $(\phi, \mathcal{S})$, where ϕ is a cyclic ordering of the edges around a vertex (also known as the *rotation system* of G) and $\mathcal{S} = (S_1, S_2, \ldots, S_{2g})$ is the cyclic ordering of the directed sides of the polygon. Each S_i is an ordered sequence of the vertices, from the tail to the head of the side S_i. Moreover every S_i is paired with some other side, say S_i^{-1} in $\mathcal{S}$, such that the jth vertex of S_i (say from the tail of S_i) is the same as the jth vertex of S_i^{-1} (from the tail of S_i^{-1}).

2.2 Complexity Theory

For a nondeterministic machine M, let $acc_M(x)$ and $rej_M(x)$ denote the number of accepting computations and the number of rejecting computations respectively on an input x. Denote $gap_M(x) = acc_M(x) - rej_M(x)$.

▶ **Definition 3.** A language L is in SPL if there exists a logspace bounded nondeterministic machine M so that for all inputs x, $gap_M(x) \in \{0, 1\}$ and $x \in L$ if and only if $gap_M(x) = 1$. $\mathsf{FL}^{\mathsf{SPL}}$ is the class of functions computed by a logspace machine with an SPL oracle. UL is the class of languages L, decided by a nondeterministic logspace machine (say M), such that for every string in L, M has exactly one accepting path and for a string not in L, M has no accepting path.

Alternatively, we can define SPL as the class of problems logspace reducible to the problem of checking whether the determinant of a matrix is 0 or not under the promise that the determinant is either 0 or 1. For definitions of other complexity classes refer to any standard textbooks such as [2, 17]. All reductions discussed in this paper are logspace reductions.

Given an undirected graph $G = (V, E)$, a *matching* M is a subset of E such that no two edges in M have a vertex in common. A *maximum matching* is a matching of maximum cardinality. M is said to be a perfect matching if every vertex is an endpoint of some edge in M.

▶ **Definition 4.** We define the following computational problems related to matching:
- Decision-BPM : Given a bipartite graph G, checking if G has a perfect matching.
- Search-BPM: Given a bipartite graph G, constructing a perfect matching, if one exists.
- Unique-BPM: Given a bipartite graph G, checking if G has a unique perfect matching.

2.3 Necessary Prior Results

▶ **Lemma 5** ([6]). *For any bipartite graph G and a weight function w, if all circulations of G are non-zero, then G has a unique minimum weight perfect matching.*

▶ **Lemma 6** ([1]). *For any weighted graph G assume that the minimum weight perfect matching in G is unique and also for any subset of edges $E' \subseteq E$, the minimum weight perfect matching in $G \setminus E'$ is also unique. Then deciding if G has a perfect matching is in* SPL*. Moreover, computing the perfect matching (in case it exists) is in* $\mathsf{FL}^{\mathsf{SPL}}$.

3 Embedding on a Grid

We define k-ori-GG to be the class of genus g graphs such that: for every $G \in$ k-ori-GG, G is a grid graph embedded on a grid of size $2m \times 2m$. We assume that the distance between adjacent horizontal (and similarly vertical) vertices is of unit length. The entire boundary of the grid is divided into $4g$ segments, and each segment has even length, for some constant g. The $4g$ segments are labeled as $(S_1, S_2, S'_1, S'_2, \ldots S_{2i-1}, S_{2i}, S'_{2i-1}, S'_{2i},$

$\ldots, S_{2g-1}, S_{2g}, S'_{2g-1}, S'_{2g})$, together with a direction, namely, S_i is directed from counter-clockwise and S'_i is directed from clockwise for each $i \in [2g]$. The jth vertex on a segment S_i is the jth vertex on the border of the grid, starting from the tail of the segment S_i and going along the direction of the segment. Finally the segments S_i and S'_i are glued to each other for each $i \in [2g]$ in the same direction. In other words, the jth vertex on segment S_i is the same as the jth vertex on segment S'_i. Also there are no edges along the boundary of the grid. In Theorem 7 we show that it is enough to consider graphs in K-ORI-GG.

▶ **Theorem 7.** *Given a 2-cell embedding of a graph G of constant genus, there is a logspace transducer that constructs a graph $G' \in$ K-ORI-GG, such that, there is a perfect matching in G iff there is a perfect matching in G'. Moreover, given a perfect matching M' in G', in logspace one can construct a perfect matching M in G.*

We divide the construction in Theorem 7 in an iterative manner starting from a 2-cell embedding. Applying Lemma 8 we first get an embedding on the polygonal schema of the graph. Then we normalize the obtained polygonal schema by applying Theorem 9. Finally we give an embedding of the graph on a grid by applying Lemma 10.

▶ **Lemma 8.** *Given the combinatorial embedding of a constant genus graph we can find a polygonal schema for the graph in logspace.*

▶ **Theorem 9.** *Given a combinatorial embedding of constant genus, say g (which is positive or otherwise), for a graph G, in logspace we can find a polygonal schema for the graph in normal form. of genus $O(|g|)$ in magnitude, and also the corresponding combinatorial embedding.*

Let K-GON-BI be the class of constant genus, bipartite graphs along with an embedding given on the polygonal schema in normal form of the surface in which the graph has an embedding. Moreover, for every graph in this class, no edge has both its end points incident on the boundary of the polygon.

▶ **Lemma 10.** *If G is an orientable graph in* K-GON-BI, *then one can get a logspace, matching-preserving reduction form G to a graph $H \in$* K-ORI-GG

4 New Upper Bounds

In this section we establish new upper bounds on the space complexity of certain matching problems on bipartite constant genus graphs, embedded on a 'genus g grid'.

▶ **Definition 11.** If C is a cycle in G, we denote the circulation of C with respect to a weight function w as $circ_w(C)$. For any subset $E' \subseteq C$, $circ_w(E')$ is the value of the circulation restricted to the edges of E'. An example of a cycle on a grid is given in Figure 2.

▶ **Theorem 12** (Main Theorem). *Given a graph $G \in$ K-ORI-GG, there exists a logspace computable and polynomially bounded weight function $W : E(G) \to \mathbb{Z}$, such that for any cycle $C \in G$, $circ_W(C) \neq 0$.*

▶ **Theorem 13.** *For a graph embedded on a constant genus surface,*

(a) DECISION-BPM *is in* SPL,
(b) SEARCH-BPM *is in* $\mathsf{FL}^{\mathsf{SPL}}$ *and*
(c) UNIQUE-BPM *is in* SPL.

Proof. As a result of Theorem 7, we can assume that our input graph $G \in$ K-ORI-GG. Using Theorem 12 and Lemma 5 we get a logspace computable weight function W, such that the minimum weight perfect matching in G with respect to W is unique. Moreover, for any subset $E' \subseteq E$, Theorem 12 is valid for the subgraph $G \setminus E'$ also, with respect to the same weight function W. Now (a) and (b) follows from Lemma 6. Checking for uniqueness can be done by first computing a perfect matching, then deleting an edge from the matching and rechecking to see if a perfect matching exists in the new graph. If it does, then G did not have a unique perfect matching, else it did. Note that Theorem 12 is valid for any graph formed by deletion of edges of G. ◀

Theorem 12 also gives an alternative proof of directed graph reachability for constant genus graphs.

▶ **Theorem 14** ([4, 11]). *Directed graph reachability for constant genus graphs is in* UL.

The proof of Theorem 14 follows from Lemma 15 and [4]. We adapt Lemma 15 from [6].

▶ **Lemma 15.** *There exist a logspace computable weight function that assigns polynomially bounded weights to the edges of a directed graph such that: (a) the weights are skew symmetric, i.e., w(u,v) = - w(v,u), and (b) the sum of weights along any (simple) directed cycle is non-zero.*

▶ **Lemma 16.** *In any class of graphs closed under the subdivision of edges,* Theorem 12 *implies the hypothesis of* Lemma 15.

4.1 Proof of Main Theorem

Proof of Theorem 12. For a graph $G \in$ K-ORI-GG, we define W is a linear combination of the following $4g+1$ weight functions defined below. This is possible in logspace since g is constant.

Define $4g+1$ weight functions as follows:

- For each $i \in [2g]$,

$$w_i(e) = \begin{cases} 1 & \text{if } e \text{ lies on the segment } S_i \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

- For each $i \in [2g]$,

$$w'_i(e) = \begin{cases} j & \text{if } e \text{ lies on the segment } S_i \text{ at index } j \text{ from the head of } S_i \text{ and } j \text{ is odd} \\ -j & \text{if } e \text{ lies on the segment } S_i \text{ at index } j \text{ from the head of } S_i \text{ and } j \text{ is even} \\ 0 & \text{otherwise} \end{cases} \tag{4.2}$$

-

$$w''(e) = \begin{cases} (-1)^{i+j}(i-1) & \text{if } e \text{ is the } j\text{th horizontal edge from left, lying in row } i \\ & \text{from bottom, and not lying on the boundary} \\ 0 & \text{otherwise} \end{cases} \tag{4.3}$$

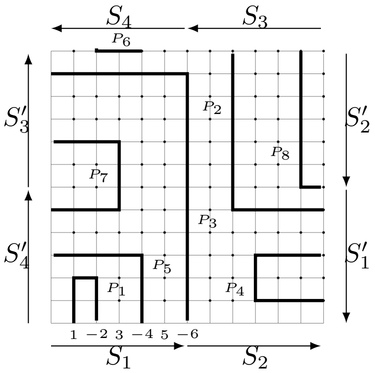


Figure 2 Example of a cycle on the grid that crosses each segment an even number of times with the weights w'_1

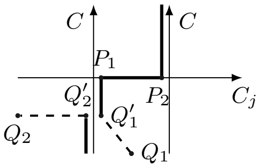


Figure 3 Construction of a path from Q_1 to Q_2 in $\Gamma \setminus C$ (the dotted path is a path between Q_1 and Q'_1 (resp. between Q_2 and Q'_2).

Note that if e does not lie on the boundary of the grid then $w''(e)$ is same as the weight function defined in [6].

Let C be a simple cycle in G. If C does not intersect any of the boundary segments, then C does not have any edge on the boundary since there are no edges along the boundary by definition of K-ORI-GG. Therefore $circ_{w''}(C) \neq 0$ by [6]. Now suppose there exists a segment S_i, such that C crosses S_i an odd number of times. Then $circ_{w_i}(C) \neq 0$. Otherwise C crosses each segment an even number of times. Now without loss of generality, assume C intersects segment S_1. Let E_1^C be the set of edges of C that intersect S_1. Note that $circ_{w'_1}(C) = circ_{w'_1}(E_1^C)$. By Lemma 18 it follows that the edges of E_1^C, alternate between going out and coming into the grid. Then using Lemma 19 we get that $circ_{w'_1}(E_1^C) \neq 0$ and thus $circ_{w'_1}(C) \neq 0$. (See below for Lemma 18 and 19) ◀

To establish Lemma 18 we use an argument (Lemma 17) from homology theory. For two cycles (directed or undirected) C_1 and C_2, let $I(C_1, C_2)$ denote the number of times C_1 and C_2 cross each other (that is one of them goes from the left to the right side of the other, or vice versa).

Next we adapt the following Lemma from Cabello and Mohar [5]. Here we assume we are given an orientable surface (Cabello and Mohar gives a proof for a graph on a surface).

▶ **Lemma 17** ([5]). *Given a genus g orientable, surface Γ, let $\mathcal{C} = \{C_i\}_{i \in [2g]}$ be a set of cycles that generate the first homology group $H_1(\Gamma)$. A cycle C in Γ is non-separating if and only if there is some cycle $C_i \in \mathcal{C}$ such that $I(C, C_i) \equiv 1(\text{mod}2)$.*

Proof. Let $\tilde{C}$ be some cycle in Γ. We can write $\tilde{C} = \sum_{i \in [2g]} t_i C_i$ since $\mathcal{C}$ generates $H_1(\Gamma)$. Define $I_{\tilde{C}}(C) = \sum_{i \in [2g]} t_i I(C, C_i)(\mod 2)$. One can verify that $I_{\tilde{C}} : \mathcal{C}_1(\Gamma) \to \mathbb{Z}_2$ is a group homomorphism. Now since $\mathcal{B}_1(\Gamma)$ is a normal subgroup of $\mathcal{B}_1(\Gamma)$, $I_{\tilde{C}}$ induces a homomorphism from $H_1(\Gamma)$ to $\mathbb{Z}_2$.

Any cycle is separating if and only if it is homologous to the empty set. Therefore if C is separating, then $C \in \mathcal{B}_1(\Gamma)$ and thus every homomorphism from $H_1(\Gamma)$ to $\mathbb{Z}_2$ maps it to 0. Hence for every $i \in [2g]$, $I(C, C_i) \equiv I_{C_i}(C) = 0$.

Suppose C is non-separating. One can construct a cycle C' on Γ, that intersects C exactly once. Let $C' = \sum_{i \in [2g]} t'_i C_i$. Now $1 \equiv I_{C'}(C) \equiv \sum_{i \in [2g]} t'_i I(C, C_i)(\text{mod}2)$. This implies that there exists $i \in [2g]$ such that $I(C, C_i) \equiv 1(\text{mod}2)$. ◀

▶ **Lemma 18.** *Let C be a simple directed cycle on a genus g orientable surface Γ and let $\mathcal{C} = \{C_i\}_{i \in [2g]}$ be a system of $2g$ directed cycles on Γ, having exactly one point in common and $\Gamma \setminus \mathcal{C}$ is the fundamental polygon, say Γ'. If $I(C, C_i)$ is even for all $i \in [2g]$ then for all $j \in [2g]$, C alternates between going from left to right and from right to left of the cycle C_j in the direction of C_j (if C crosses C_j at all).*

Proof. Suppose there exists a $j \in [2g]$ such that C does not alternate being going from left to right and from right to left with respect to C_j. Thus if we consider the ordered set of points where C intersects C_j, ordered in the direction of C_j, there are two consecutive points (say P_1 and P_2) such that at both these points C crosses C_j in the same direction.

Let Q_1 and Q_2 be two points in $\Gamma \setminus C$. We will show that there exists a path in $\Gamma \setminus C$ between Q_1 and Q_2. Consider the shortest path from Q_1 to C. Let Q'_1 be the point on this path that is as close to C as possible, without lying on C. Similarly define a point Q'_2 corresponding to Q_2. Note that it is sufficient for us to construct a path between Q'_1 and Q'_2 in $\Gamma \setminus C$. If both Q'_1 and Q'_2 locally lie on the same side of C, then we get a path from Q'_1 to Q'_2 not intersecting C, by traversing along the boundary of C. Now suppose Q'_1 and Q'_2 lie on opposite sides (w.l.o.g. assume that Q'_1 lies on the right side) of C. From Q'_1 start traversing the cycle until you reach cycle C_j (point P_1 in Figure 3). Continue along cycle C_j towards the adjacent intersection point of C and C_j, going as close to C as possible, without intersecting it (point P_2 in Figure 3). Essentially this corresponds to switching from one side of C to the other side without intersecting it. Next traverse along C to reach Q'_2. Thus we have a path from Q'_1 to Q'_2 in $\Gamma \setminus C$. We give an example of this traversal in Figure 3. This implies that C is non-separating.

It is well known that $\mathcal{C}$ forms a generating set of $H_1(\Gamma)$, the first homology group of the surface. Now from Lemma 17 it follows that $I(C, C_l) \equiv 1(\text{mod}2)$ for some $l \in [2g]$, which is a contradiction. ◀

▶ **Lemma 19.** *Let G be a graph in* K-ORI-GG *with C being a simple cycle in G and E_1^C being the set of edges of C that intersect segment S_1. Assume $|E_1^C|$ is even and the edges in E_1^C alternate between going out and coming into the grid. Let $i_1 < i_2 < \ldots < i_{2p-1} < i_{2p}$ be the distinct indices on S_1 where C intersects it. Then, $\left|circ_{w'_1}(E_1^C)\right| = |\sum_{k=1}^{p}(i_{2k} - i_{2k-1})|$ and thus non-zero unless E_1^C is empty.*

Proof. Let $e_j = (u_j, v_j)$ for $j \in [2p]$ be the $2p$ edges of G incident on the segment S_1. Assume without loss of generality that the vertices v_j's lie on S_1. Assign an orientation to C such that e_1 is directed from u_1 to v_1. Also assume that i_1 is even and the circulation gives a positive sign to the edge e_1. Therefore $circ_{w'_1}(\{e_1\}) = -i_1$.

Now consider any edge e_j such that j is even. By Lemma 18, the edge enters the segment S_1 (i.e., the head of the edge with respect to the assigned orientation is incident on S_1). Suppose i_j is odd. Then consider the following cycle C' formed by tracing C from u_j to u_1, without the edges e_1 and e_j and then moving along the segment S_1 back to u_j. Since i_j is odd therefore the latter part of C' has odd length. Note that C' need not be a simple cycle. By Lemma 20, $|C'|$ is even, therefore the part of C' from u_1 to u_j also has odd length. This implies that the circulation gives a positive sign to the edge e_j. Therefore, $circ_{w_1'}(\{e_j\}) = i_j$. Similarly, if i_j is odd, then the part of C' from u_1 to u_j will have even length. Thus the circulation gives a negative sign to the edge e_j and therefore $circ_{w_1'}(\{e_j\}) = -(-i_j) = i_j$.

If j is odd, the above argument can be applied to show that $circ_{w_1'}(\{e_j\}) = -i_j$. Therefore we have, $circ_{w_1'}(E_1^C) = \sum_{k=1}^{p}(i_{2k} - i_{2k-1})$.

Now removing the assumptions at the beginning of this proof would show that the LHS and RHS of the above equation is true modulo absolute value as required. ◀

To prove Lemma 19 we need to argue that any graph in a“genus g grid" is bipartite and thus any cycle will have even length. Lemma 20 establishes this fact.

▶ **Lemma 20.** *Any graph $G \in$ k-ori-GG is bipartite.*

It is interesting to note here that similar method does not show that bipartite matching in non-orientable constant genus graphs is in SPL. The reason is that Lemma 18 crucially uses the fact that the surface is orientable. In fact, one can easily come with counterexample to the Lemma if the surface is non-orientable.

5 Reducing the non-orientable case to the orientable case

Let G be a bipartite graph embedded on a genus g non-orientable surface. As a result of Theorem 9 we can assume that we are given a combinatorial embedding (say Π) of G on a (non-orientable) polygonal schema, say $\Lambda(\Gamma)$, in the normal form with $2g'$ sides. (Here g' is a function of g.)

Let $Y = (X_1, X_2)$ be the cyclic ordering of the labels of the sides of $\Lambda(\Gamma)$, where X_2 is the ‘orientable part’ and X_1 is the ‘non-orientable part’. More precisely, for the polygonal schema in the normal form, we have: X_1 is either (σ, σ) (thus corresponds to the projective plane) or it is $(\sigma, \tau, \bar{\sigma}, \tau)$ (thus corresponds to the Klein bottle). See Figure 4.

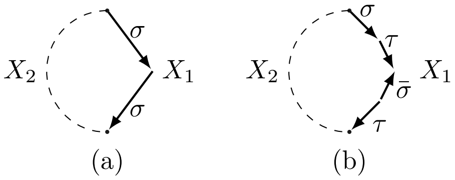


Figure 4 (a) $\Lambda(\Gamma)$ when the surface is a sum of an orientable surface and the projective plane. **(b)** $\Lambda(\Gamma)$ when the surface is a sum of an orientable surface and the Klein bottle.

Now let G be a bipartite graph embedded on a non-orientable polygonal schema $\Lambda(\Gamma)$ with $2g'$ sides. We will construct a graph G' embedded on an *orientable* polygonal schema with $4g'-2$ sides such that G has a perfect matching iff G' has a perfect matching. Moreover, given a perfect matching in G' one can retrieve in logspace a perfect matching in G. This is illustrated in Theorem 21.

▶ **Theorem 21.** *Let G be a bipartite graph given with its embedding on a non-orientable polygonal schema in normal form $\Lambda(\Gamma)$, with $2g'$ sides as above. One can construct in logspace, another graph G' together with its embedding on the polygonal schema of an orientable surface Γ' of genus $4g'-2$ such that: G has a perfect matching iff G' has a perfect matching. Moreover, given a perfect matching in G', one can construct in logspace a perfect matching in G.*

Thus we see that the non-orientable case can be reduced to the orientable case. The resulting polygonal schema need not be in the normal form. Once again we apply Theorem 9 to get a combinatorial embedding on a polygonal schema in the normal form.

Acknowledgment

The third author would like to thank Prof. Mark Brittenham from the Mathematics department at the University of Nebraska-Lincoln, for numerous discussions that they had and for providing valuable insight into topics in algebraic topology. We would also like to thank the anonymous referees for their valuable comments and suggestions which helped in improving the overall presentation of the paper.

References

1 Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting: Uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59:164–181, 1999.

2 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 1 edition, 2009.

3 V. Arvind and Partha Mukhopadhyay. Derandomizing the isolation lemma and lower bounds for circuit size. In *Proceedings of RANDOM '08*, pages 276–289, 2008.

4 Chris Bourke, Raghunath Tewari, and N. V. Vinodchandran. Directed planar reachability is in unambiguous log-space. *ACM Trans. Comput. Theory*, 1(1):1–17, 2009.

5 Sergio Cabello and Bojan Mohar. Finding shortest non-separating and non-contractible cycles for topologically embedded graphs. *Discrete Comput. Geom.*, 37(2):213–235, 2007.

6 Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory of Computing Systems*, 47(3):737–757, 2010.

7 Samir Datta, Raghav Kulkarni, Raghunath Tewari, and N. V. Vinodchandran. Space complexity of perfect matching in bounded genus bipartite graphs. Technical Report TR10-079, Electronic Colloquium on Computational Complexity, 2010.

8 Anna Gal and Avi Wigderson. Boolean complexity classes vs. their arithmetic analogs. *Random Structures and Algorithms*, 9:1–13, 1996.

9 Thanh Minh Hoang. On the matching problem for special graph classes. In *IEEE Conference on Computational Complexity*, pages 139–150, 2010.

10 Raghav Kulkarni, Meena Mahajan, and Kasturi R. Varadarajan. Some perfect matchings and perfect half-integral matchings in NC. *Chicago Journal of Theoretical Computer Science*, 2008(4), September 2008.

11 Jan Kynčl and Tomáš Vyskočil. Logspace reduction of directed reachability for bounded genus graphs to the planar case. *ACM Trans. Comput. Theory*, 1(3):1–11, 2010.

12 William S. Massey. *A Basic Course in Algebraic Topology.* Springer-Verlag, 1991.

13 Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces.* John Hopkins University Press, 2001.

14 Ketan Mulmuley, Umesh Vazirani, and Vijay Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

15 Klaus Reinhardt and Eric Allender. Making nondeterminism unambiguous. *SIAM Journal of Computing*, 29:1118–1131, 2000. An earlier version appeared in FOCS 1997, pp. 244–253.

16 C. Thomassen. The graph genus problem is np-complete. *J. Algorithms*, 10(4):568–576, 1989.

17 Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach.* Springer-Verlag, 1999.

The Recognition of Triangle Graphs

George B. Mertzios[1]

1 Department of Computer Science, Technion,
Haifa, Israel*
mertzios@cs.technion.ac.il

Abstract

Trapezoid graphs are the intersection graphs of trapezoids, where every trapezoid has a pair of opposite sides lying on two parallel lines L_1 and L_2 of the plane. Strictly between permutation and trapezoid graphs lie the *simple-triangle* graphs – also known as *PI* graphs (for Point-Interval) – where the objects are triangles with one point of the triangle on L_1 and the other two points (i.e. interval) of the triangle on L_2, and the *triangle* graphs – also known as *PI** graphs – where again the objects are triangles, but now there is no restriction on which line contains one point of the triangle and which line contains the other two. The complexity status of both triangle and simple-triangle recognition problems (namely, the problems of deciding whether a given graph is a triangle or a simple-triangle graph, respectively) have been the most fundamental open problems on these classes of graphs since their introduction two decades ago. Moreover, since triangle and simple-triangle graphs lie naturally between permutation and trapezoid graphs, and since they share a very similar structure with them, it was expected that the recognition of triangle and simple-triangle graphs is polynomial, as it is also the case for permutation and trapezoid graphs. In this article we surprisingly prove that the recognition of triangle graphs is NP-complete, even in the case where the input graph is known to be a trapezoid graph.

1998 ACM Subject Classification F.2.2 Computations on discrete structures, G.2.2 Graph theory

Keywords and phrases Intersection graphs, trapezoid graphs, PI graphs, PI* graphs, recognition problem, NP-complete

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.591

1 Introduction

A graph $G = (V, E)$ with n vertices is the *intersection graph* of a family $F = \{S_1, \ldots, S_n\}$ of subsets of a set S if there exists a bijection $\mu : V \to F$ such that for any two distinct vertices $u, v \in V$, $uv \in E$ if and only if $\mu(u) \cap \mu(v) \neq \emptyset$. Then, F is called an *intersection model* of G. Note that every graph has a trivial intersection model based on adjacency relations [18]. However, some intersection models provide a natural and intuitive understanding of the structure of a class of graphs, and turn out to be very helpful to obtain structural results, as well as to find efficient algorithms to solve optimization problems [18]. Many important graph classes can be described as intersection graphs of set families that are derived from some kind of geometric configuration.

Consider two parallel horizontal lines on the plane, L_1 (the upper line) and L_2 (the lower line). Various intersection graphs can be defined on objects formed with respect to these two lines. In particular, for *permutation* graphs, the objects are line segments that have one

* Current Address: Caesarea Rothschild Institute for Computer Science, University of Haifa, Haifa, Israel.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 591–602
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

endpoint on L_1 and the other one on L_2. Generalizing to objects that are trapezoids with one interval on L_1 and the opposite interval on L_2, *trapezoid* graphs have been introduced independently in [5] and [6]. Given a trapezoid graph G, an intersection model of G with trapezoids between L_1 and L_2 is called a *trapezoid representation* of G. Trapezoid graphs are perfect graphs [3, 9] and generalize in a natural way both interval graphs (when the trapezoids are rectangles) and permutation graphs (when the trapezoids are trivial, i.e. lines). In particular, the main motivation for the introduction of trapezoid graphs was to generalize some well known applications of interval and permutation graphs on channel routing in integrated circuits [6].

Moreover, two interesting subclasses of trapezoid graphs have been introduced in [5]. A trapezoid graph G is a *simple-triangle* graph if it admits a trapezoid representation, in which every trapezoid is a triangle with one point on L_1 and the other two points (i.e. interval) on L_2. Similarly, G is a *triangle* graph if it admits a trapezoid representation, in which every trapezoid is a triangle, but now there is no restriction on which line between L_1 and L_2 contains one point of the triangle and which one contains the other two points (i.e. the interval) of the triangle. Such an intersection model of a simple-triangle (resp. triangle) graph G with triangles between L_1 and L_2 is called a *simple-triangle* (resp. *triangle* representation of G). Simple-triangle and triangle graphs are also known as *PI* and *PI** graphs, respectively [3–5,15], where PI stands for “Point-Interval”; note that, using this notation, permutation graphs are *PP* (for “Point-Point”) graphs, while trapezoid graphs are *II* (for “Interval-Interval”) graphs [5]. In particular, both interval and permutation graphs are strictly contained in simple-triangle graphs, which are strictly contained in triangle graphs, which are strictly contained in trapezoid graphs [3, 5].

Due to both their interesting structure and their practical applications, trapezoid graphs have attracted many research efforts. In particular, efficient algorithms for several optimization problems that are NP-hard in general graphs have been designed for trapezoid graphs [2, 7, 10, 12, 13, 16, 25], which also apply to triangle and simple-triangle graphs. Furthermore, several efficient algorithms appeared for the recognition problems of both permutation [9,17] and trapezoid graphs [14,16,21]; see [26] for an overview.

In spite of this, the complexity status of both triangle and simple-triangle recognition problems have been the most fundamental open problems on these classes of graphs since their introduction two decades ago [3]. Since, on the one hand, very few subclasses of perfect graphs are known to be NP-hard to recognize (for instance, *perfectly orderable* graphs [23], *EPT* graphs [11], and recently *tolerance* and *bounded tolerance* graphs [22]) and, on the other hand, triangle and simple-triangle graphs lie naturally between permutation and trapezoid graphs, while they share a very similar structure with them, it was expected that the recognition of triangle and simple-triangle graphs was polynomial.

Our contribution

In this article we establish the complexity of recognizing triangle graphs. Namely, we prove that this problem is surprisingly NP-hard, by providing a reduction from the 3SAT problem. Specifically, given a boolean formula formula ϕ in conjunctive normal form with three literals in every clause (3-CNF), we construct a trapezoid graph G_ϕ, which is a triangle graph if and only if ϕ is satisfiable. Therefore, as the recognition problems for both triangle and simple-triangle graphs are in the complexity class NP, it follows in particular that the triangle graph recognition problem is NP-complete. This complements the recent surprising result that the recognition of *parallelogram* graphs (i.e. the intersection graphs of parallelograms between two parallel lines L_1 and L_2), which coincides with *bounded tolerance* graphs, is NP-complete [22].

Organization of the paper.

Background definitions and properties of trapezoid graphs and their representations are presented in Section 2. In Section 3 we introduce the notion of a *standard trapezoid representation*, the existence of which is a sufficient condition for a trapezoid graph to be a triangle graph. In Sections 4 and 5, we investigate the structure of some specific trapezoid and triangle graphs, respectively, and prove special properties of them. We use these graphs as parts of the gadgets in our reduction of 3SAT to the recognition problem of triangle graphs, which we present in Section 6. Finally, we discuss the presented results and further research in Section 7. Due to space limitations, some proofs are omitted; a full version can be found in [19].

2 Triangle and simple-triangle graphs

In this section we provide some notation and properties of trapezoid graphs and their representations, which will be mainly applied in the sequel to triangle and simple-triangle graphs.

Notation. We consider in this article simple undirected and directed graphs with no loops or multiple edges. In an undirected graph G, the edge between vertices u and v is denoted by uv, and in this case u and v are said to be *adjacent* in G. Given a graph $G = (V, E)$ and a subset $S \subseteq V$, $G[S]$ denotes the induced subgraph of G on the vertices in S. Furthermore, we denote for simplicity by $G-S$ the induced subgraph $G[V \setminus S]$ of G. Moreover, given a graph G, we denote its vertex set by $V(G)$. A connected graph $G = (V, E)$ is called *k-connected*, where $k \geq 1$, if k is the smallest number of vertices that have to be removed from G such that the resulting graph is disconnected. Furthermore, a vertex v of a 1-connected graph G is called a *cut vertex* of G, if $G - \{v\}$ is disconnected. By possibly performing a small shift of the endpoints, we assume throughout the article without loss of generality that all endpoints of the trapezoids (resp. triangles) in a trapezoid (resp. triangle or simple-triangle) representation are distinct [8, 10, 12]. Given a trapezoid (resp. triangle or simple-triangle) graph G along with a trapezoid (resp. triangle or simple-triangle) representation R, we may not distinguish in the following between a vertex of G and the corresponding trapezoid (resp. triangle) in R, whenever it is clear from the context. Moreover, given an induced subgraph H of G, we denote by $R[H]$ the restriction of the representation R on the trapezoids (resp. triangles) of H.

Consider a trapezoid graph $G = (V, E)$ and a trapezoid representation R of G, where for any vertex $u \in V$ the trapezoid corresponding to u in R is denoted by T_u. Since trapezoid graphs are also cocomparability graphs (there is a transitive orientation of the complement) [9], we can define the partial order $(V, \ll_R)$, such that $u \ll_R v$, or equivalently $T_u \ll_R T_v$, if and only if T_u lies completely to the left of T_v in R (and thus also $uv \notin E$). Otherwise, if neither $T_u \ll_R T_v$ nor $T_v \ll_R T_u$, we will say that T_u *intersects* T_v in R (and thus also $uv \in E$). Furthermore, we define the total order $<_R$ on the lines L_1 and L_2 in R as follows. For two points a and b on L_1 (resp. on L_2), if a lies to the left of b on L_1 (resp. on L_2), then we will write $a <_R b$.

There are several trapezoid representations of a particular trapezoid graph G. For instance, given one such representation R, we can obtain another one R' by *vertical axis flipping* of R, i.e. R' is the mirror image of R along an imaginary line perpendicular to L_1 and L_2. Moreover, we can obtain another representation R'' of G by *horizontal axis flipping* of R, i.e. R'' is the mirror image of R along an imaginary line parallel to L_1 and L_2. We will

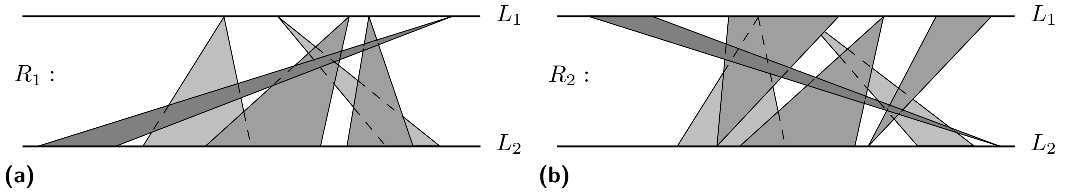


Figure 1 (a) A simple-triangle representation R_1 and (b) a triangle representation R_2.

use extensively these two basic operations throughout the article. For every trapezoid T_u in R, where $u \in V$, we define by $l(u)$ and $r(u)$ (resp. $L(u)$ and $R(u)$) the lower (resp. upper) left and right endpoint of T_u, respectively (cf. the trapezoid T_v in Figure 2). Since every triangle and simple-triangle representation is a special type of a trapezoid representation, all the above notions can be also applied to triangle and simple-triangle graphs. Note here that, if R is a simple-triangle representation of $G = (V, E)$, then $L(u) = R(u)$ for every $u \in V$; similarly, if R is a triangle representation of G, then $L(u) = R(u)$ or $l(u) = r(u)$ for every $u \in V$. An example of a simple-triangle and a triangle representation is shown in Figure 1.

It can be easily seen that every triangle (resp. single-triangle) graph G has a triangle (resp. single-triangle) representation of G, in which the endpoints of the triangles in both lines L_1 and L_2 are integers. That is, every triangle (resp. single-triangle) graph G with n vertices has a representation with size polynomial on n, and thus the recognition problems of both both triangle and simple-triangle graphs are in NP, as the next observation states.

▶ Observation 1. The triangle and simple-triangle graph recognition problems are in the complexity class NP.

3 Standard trapezoid representations

In this section we investigate several properties of trapezoid and triangle graphs and their representations. In particular, we introduce the notion of a *standard trapezoid representation*. We prove that a sufficient condition for a trapezoid graph G to be a triangle graph is that G admits such a standard representation. These properties of trapezoid and triangle graphs, as well as the notion of a standard trapezoid representation will then be used in our reduction for the triangle graph recognition problem. In order to define the notion of a standard trapezoid representation (cf. Definition 3), we first provide the following two definitions regarding an arbitrary trapezoid T_v in a trapezoid representation.

▶ **Definition 1.** Let R be a trapezoid representation of a trapezoid graph $G = (V, E)$ and T_v be a trapezoid in R, where $v \in V$. Let R' and R'' be the representations obtained by vertical axis flipping and by horizontal axis flipping of R, respectively. Then,

- T_v is *upper-right-closed in* R if there exist two vertices $u, w \in N(v)$, such that $T_u \ll_R T_w$, $L(w) <_R R(v)$, and $r(v) <_R l(w)$; otherwise T_v is *upper-right-open in* R,
- T_v is *upper-left-closed in* R if T_v is upper-right-closed in R'; otherwise T_v is *upper-left-open in* R,
- T_v is *lower-right-closed in* R if T_v is upper-right-closed in R''; otherwise T_v is *lower-right-open in* R,
- T_v is *lower-left-closed in* R if T_v is lower-right-closed in R'; otherwise T_v is *lower-left-open in* R.

▶ **Definition 2.** Let R be a trapezoid representation of a trapezoid graph $G = (V, E)$ and T_v be a trapezoid in R, where $v \in V$. Then,

- T_v is *right-closed in* R if T_v is both upper-right-closed and lower-right-closed in R; otherwise T_v is *right-open in* R,
- T_v is *left-closed in* R if T_v is both upper-left-closed and lower-left-closed in R; otherwise T_v is *left-open in* R,
- T_v is *closed* in R if T_v is both right-closed and left-closed in R; otherwise T_v is *open* in R.

As an example for Definitions 1 and 2, consider the trapezoid representation R in Figure 2. In this figure, the trapezoid T_v is upper-left-closed and lower-left-closed, as well as upper-right-closed and lower-right-open. Therefore, T_v is left-closed and right-open in R, i.e. T_v is open in R. For better visibility, we place in Figure 2 three bold bullets on the upper right, upper left, and lower left endpoints of the trapezoid T_v, in order to indicate that T_v is upper-right-closed, upper-left-closed, and lower-left-closed, respectively.

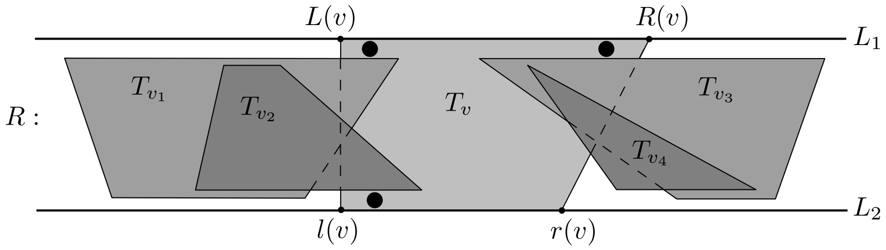


Figure 2 A standard trapezoid representation R, in which the trapezoid T_v is left-closed, upper-right-closed, and lower-right-open.

We are now ready to define the notion of a standard trapezoid representation.

▶ **Definition 3.** Let $G = (V, E)$ be a trapezoid graph and R be a trapezoid representation of G. If, for every $v \in V$, the trapezoid T_v is open in R or T_v is a triangle in R, then R is a *standard trapezoid representation.*

For example, the trapezoid representation R in Figure 2 is a standard. Indeed, none of the trapezoids $T_{v_1}, T_{v_2}, T_{v_3}$ is right-closed or left-closed, while T_v is lower-right-open (and therefore also right-open by Definition 2). Thus, each of the trapezoids T_v, T_{v_1}, T_{v_2}, and T_{v_3} is open in R. Moreover, T_{v_4} is a triangle in R.

Note that every triangle representation is a standard trapezoid representation by Definition 3. We now provide the main theorem of this section, which states a sufficient condition for a trapezoid graph to be triangle.

▶ **Theorem 4.** *Let $G = (V, E)$ be a trapezoid graph. If there exists a standard trapezoid representation of G, then G is a triangle graph.*

4 Basic constructions of trapezoid graphs

In this section we investigate some small trapezoid graphs and prove special properties of them. These graphs will then be used as parts of the gadgets in our reduction of 3SAT to the recognition problem of triangle graphs in Section 6. For simplicity of the presentation, we do not distinguish in the sequel of the article between a vertex v of a trapezoid graph G and the trapezoid T_v of v in a trapezoid representation of G.

▶ **Lemma 5.** *Let $G = (V, E)$ be the trapezoid graph induced by the trapezoid representation of Figure 3a. Then, in* any *trapezoid representation R of G, such that $v \ll_R v'$,*

- *v is upper-right-closed in R and v' is lower-left-closed in R, or*
- *v is lower-right-closed in R and v' is upper-left-closed in R.*

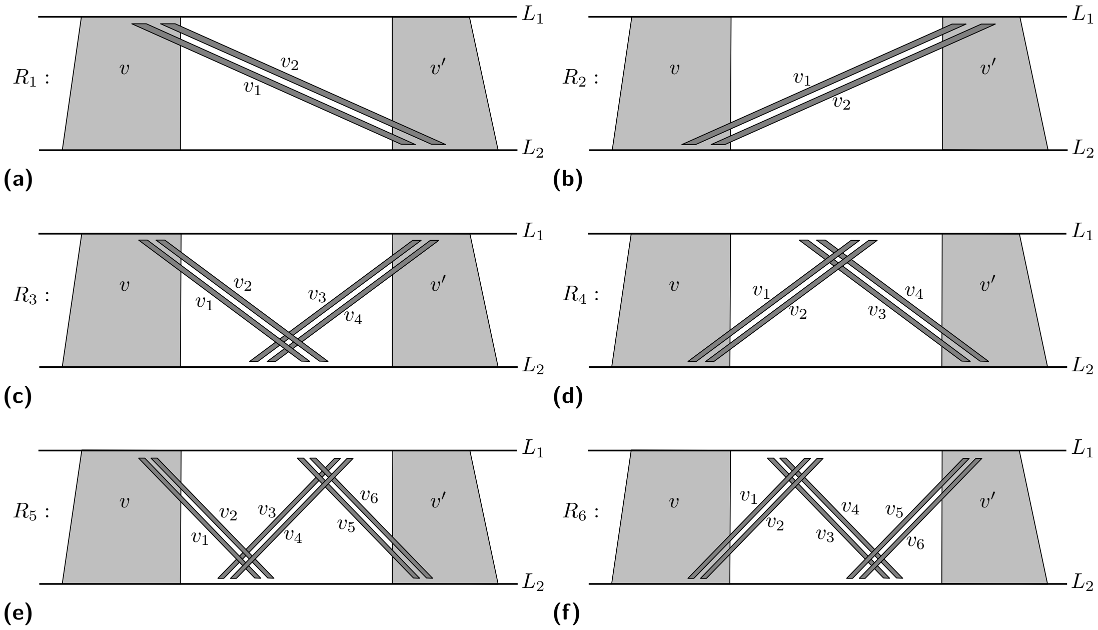


Figure 3 Six basic trapezoid representations.

The next two lemmas concern similar properties of the graphs induced by the trapezoid representations of Figures 3c and 3e, respectively.

▶ **Lemma 6.** *Let $G = (V, E)$ be the trapezoid graph induced by the trapezoid representation of Figure 3c. Then, in* any *trapezoid representation R of G, such that $v \ll_R v'$,*

- *v is upper-right-closed in R and v' is upper-left-closed in R, or*
- *v is lower-right-closed in R and v' is lower-left-closed in R.*

▶ **Lemma 7.** *Let $G = (V, E)$ be the trapezoid graph induced by the trapezoid representation of Figure 3e. Then, in* any *trapezoid representation R of G, such that $v \ll_R v'$,*

- *v is upper-right-closed in R and v' is lower-left-closed in R, or*
- *v is lower-right-closed in R and v' is upper-left-closed in R.*

5 Basic constructions of triangle graphs

In this section we investigate the structure of some specific triangle graphs and devise special properties of them. As triangle graphs are also trapezoid graphs, in order to prove these properties, we use some of the results provided in Section 4. Similarly to the trapezoid graphs investigated in Section 4, also the investigated graphs of the present section will then be used as gadgets in our reduction for the triangle graph recognition problem in Section 6. Before investigating any specific triangle graph, we first provide in the next theorem a generic result that concerns the triangle representations of the 1-connected triangle graphs.

▶ **Theorem 8.** *Let $G = (V, E)$ be a 1-connected triangle graph and $v \in V$ be a cut vertex of G. Then, in* any *triangle representation R of G, the trapezoid of v is open in R.*

We now use the generic Theorem 8, as well as the results of Section 4, in order to prove some properties of the trapezoid representations of Figure 4. Note that, although the representations of Figure 4 are not triangle representations, they are standard trapezoid representations, and thus the graphs induced by these representations are triangle graphs by Theorem 4.

▶ **Lemma 9.** *Let $G = (V, E)$ be the triangle graph induced by the trapezoid representation of Figure 4a. Then, in* any *triangle representation R of G, such that $a_7 \ll_R u$, u is left-open in R if and only if w is right-open in R.*

Proof. Let R be a triangle representation of G, such that $a_7 \ll_R u$. Note that $G - \{u, w\}$ has the two connected components $G_1 = G[a_1, a_2, a_3, a_4, a_5, a_6, a_7]$ and $G_2 = G[v, b_1, b_2, b_3, b_4, b_5, b_6]$, and thus one of these two induced subgraphs of G lies completely to the left of the other in R. If $v \ll_R a_7 \ll_R u$, then a_7 would intersect with a triangle of G_2, which is a contradiction, since $a_7 \in V(G_1)$. Furthermore, if $a_7 \ll_R v \ll_R u$, then v would intersect with a triangle of G_1, which is a contradiction, since $v \in V(G_2)$. Therefore $a_7 \ll_R u \ll_R v$; similarly, $a_7 \ll_R w \ll_R v$. Therefore, every triangle of G_1 must lie completely to the left of every triangle of G_2 in R.

($\Rightarrow$) Suppose that u is left-open in R, i.e. u is upper-left-open or lower-left-open in R. By possibly performing a horizontal axis flipping of R, we may assume without loss of generality that u is lower-left-open in R. Consider the induced subgraphs $H_1 = G[\{a_7, a_1, a_2, u\}]$ and $H_2 = G[\{a_7, a_1, a_2, w\}]$ of G. Note that both H_1 and H_2 are isomorphic to the graph investigated in Lemma 5. Since u is assumed to be lower-left-open in R (and thus also in the restriction $R[H_1]$ of the triangle representation R), Lemma 5 implies that u is upper-left-closed and a_7 is lower-right-closed in $R[H_1]$. Therefore, a_7 is lower-right-closed also in the restriction $R[H_1 - \{u\}] = R[H_2 - \{w\}]$ of R. Thus, Lemma 5 implies that a_7 is lower-right-closed and w is upper-left-closed in the restriction $R[H_2]$ of R, and thus w is upper-left-closed in R.

Consider now the induced subgraphs $H_3 = G[\{a_7, a_3, a_4, u\}]$ and $H_4 = G[\{a_7, a_3, a_4, a_5, a_6, w\}]$ of G. Note that H_3 is isomorphic to the graph investigated in Lemma 5, while H_4 is isomorphic to the graph investigated in Lemma 6. Since u is assumed to be lower-left-open in R (and thus also in $R[H_3]$), Lemma 5 implies that u is upper-left-closed and a_7 is lower-right-closed in $R[H_3]$. Therefore, a_7 is lower-right-closed also in the restriction $R[H_3 - \{u\}] = R[H_4 - \{a_5, a_6, w\}]$ of the triangle representation R. Thus, Lemma 6 implies that a_7 is lower-right-closed and w is lower-left-closed in the restriction $R[H_4]$ of R, and thus w is lower-left-closed in R. Therefore, since w is also upper-left-closed in R by the previous paragraph, it follows that w is left-closed in R.

Recall that R is a triangle representation by assumption, and thus the restriction $R[G - \{u\}]$ is also a triangle representation. Moreover, since w is left-closed in R, it follows that w is also left-closed in $R[G - \{u\}]$. Note now that the connected graph $G - \{u\}$ satisfies the conditions of Theorem 8. Indeed, w is a cut vertex of $G - \{u\}$ and $(G - \{u\}) - \{w\}$ has the two connected components $G_1 = G[a_1, a_2, a_3, a_4, a_5, a_6, a_7]$ and $G_2 = G[v, b_1, b_2, b_3, b_4, b_5, b_6]$. Therefore, since w is left-closed in $R[G - \{u\}]$, Theorem 8 implies that w is right-open in $R[G - \{u\}]$, and thus also w is right-open in R.

($\Leftarrow$) Consider the triangle representation R' of G that is obtained by performing a vertical axis flipping of R. Note that $v \ll_{R'} w$, since $w \ll_R v$. Furthermore, note that there is a trivial automorphism of G, which maps vertex u to w, vertex a_7 to v, and the vertices $\{a_1, a_2, a_3, a_4, a_5, a_6\}$ to the vertices $\{b_1, b_2, b_3, b_4, b_5, b_6\}$. That is, the relation $a_7 \ll_R u$

in the representation R is mapped by this automorphism to the relation $v \ll_{R'} w$ in the representation R'. It follows now directly by the necessity part ($\Rightarrow$) that, if w is left-open in R', then u is right-open in R'. That is, if w is right-open in R, then u is left-open in R. ◀

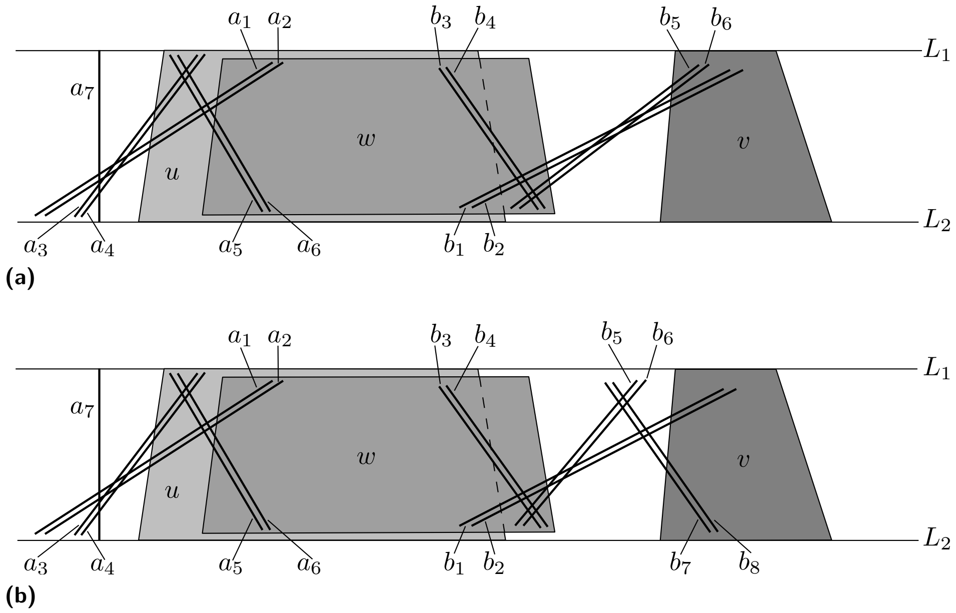


Figure 4 Two basic trapezoid representations.

Now, using Lemma 9, we can prove the next two lemmas.

▶ **Lemma 10.** *Let $G = (V, E)$ be the triangle graph induced by the trapezoid representation of Figure 4a. Then, in* any *triangle representation R of G, such that $a_7 \ll_R u$, u is left-open in R if and only if v is left-open in R.*

▶ **Lemma 11.** *Let $G = (V, E)$ be the triangle graph induced by the trapezoid representation of Figure 4b. Then, in* any *triangle representation R of G, such that $a_7 \ll_R u$, u is left-open in R if and only if v is left-closed in R.*

6 The recognition of triangle graphs

In this section we provide a reduction from the *three-satisfiability (3SAT)* problem to the problem of recognizing whether a given graph is a triangle graph. Given a boolean formula ϕ in conjunctive normal form with three literals in each clause (3-CNF), ϕ is *satisfiable* if there is a truth assignment of ϕ, such that every clause contains at least one true literal. The problem of deciding whether a given 3-CNF formula ϕ is satisfiable is one of the most known NP-complete problems. We can assume without loss of generality that each clause has literals that correspond to three distinct variables. Given the formula ϕ, we construct in polynomial time a trapezoid graph G_ϕ, such that G_ϕ is a triangle graph if and only if ϕ is satisfiable. Before constructing the whole trapezoid graph G_ϕ, we construct first some smaller trapezoid graphs for each clause and each variable that appears in the given formula ϕ.

6.1 The construction for each clause

Consider a 3-CNF formula $\phi = \alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_k$ with k clauses $\alpha_1, \alpha_2, \ldots, \alpha_k$ and n boolean variables $x_1, x_2, \ldots, x_n$, such that $\alpha_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ for $i = 1, 2, \ldots, k$. For the literals $\ell_{i,1}, \ell_{i,2}, \ell_{i,3}$ of the clause α_i, let $\ell_{i,1} \in \{x_{r_{i,1}}, \overline{x_{r_{i,1}}}\}$, $\ell_{i,2} \in \{x_{r_{i,2}}, \overline{x_{r_{i,2}}}\}$, and $\ell_{i,3} \in \{x_{r_{i,3}}, \overline{x_{r_{i,3}}}\}$, where $1 \leq r_{i,1} < r_{i,2} < r_{i,3} \leq n$. Let L_1 and L_2 be two parallel lines in the plane. For every clause α_i, where $i = 1, 2, \ldots, k$, we correspond the trapezoid representation R_{α_i} with 7 trapezoids that is illustrated in Figure 5. Note that the trapezoid of the vertex z_i in R_{α_i} is trivial, i.e. line. In this construction, the trapezoids of the vertices $v_{i,1}$, $v_{i,2}$, and $v_{i,3}$ correspond to the literals $\ell_{i,1}$, $\ell_{i,2}$, and $\ell_{i,3}$, respectively. Furthermore, by the construction of R_{α_i}, the left line of $v_{i,1}$ lies completely to the left of the left line of $v_{i,2}$ in R_{α_i}, while the left line of $v_{i,2}$ lies completely to the left of the left line of $v_{i,3}$ in R_{α_i}.

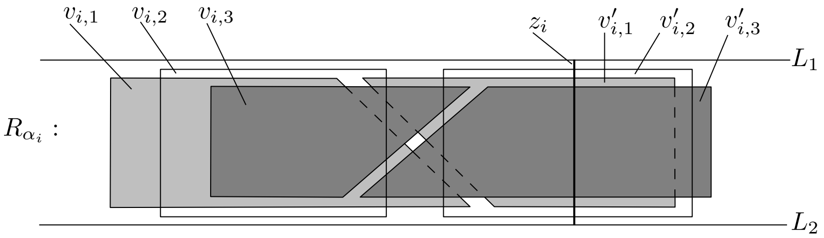


Figure 5 The construction R_{α_i} that corresponds to the clause α_i of the formula ϕ, where $i = 1, 2, \ldots, k$.

We prove now two basic properties of the construction R_{α_i} in Figure 5 for the clause α_i that will be then used in the proof of correctness of our reduction.

▶ **Lemma 12.** *Let G_{α_i} be the trapezoid graph induced by the trapezoid representation R_{α_i} of Figure 5. Then, in* any *trapezoid representation R of G_{α_i}, such that $v_{i,1} \ll_R z_i$, one of $v_{i,1}, v_{i,2}, v_{i,3}$ is right-closed in R.*

▶ **Corollary 13.** *Consider the trapezoid representation R_{α_i} of Figure 5. For every $p \in \{1, 2, 3\}$, we can locally change appropriately in R_{α_i} the right lines of $v_{i,1}, v_{i,2}, v_{i,3}$ and the left lines of $v'_{i,1}, v'_{i,2}, v'_{i,3}$, such that $v_{i,p}$ is right-closed and $v_{i,p'}$ is right-open, for every $p' \in \{1, 2, 3\} \setminus \{p\}$.*

6.2 The construction for each variable

Let x_j be a variable of the formula ϕ, where $1 \leq j \leq n$. Let x_j appear in ϕ (either as x_j or negated as $\overline{x_j}$) in the m_j clauses $\alpha_{i_{j,1}}, \alpha_{i_{j,2}}, \ldots, \alpha_{i_{j,m_j}}$, where $1 \leq i_{j,1} < i_{j,2} < \ldots < i_{j,m_j} \leq k$. Then, we correspond to the variable x_j the trapezoid representation R_{x_j} with $2m_j + 7$ trapezoids that is illustrated in Figure 6. In this construction, the trapezoids of the vertices $u_{j,t}$ and $w_{j,t}$, where $1 \leq t \leq m_j$, correspond to the appearance of the variable x_j (either as x_j or negated as $\overline{x_j}$) in the clause $\alpha_{i_{j,t}}$ in ϕ. Note that the trapezoids of the vertices $a_j^1, a_j^2, \ldots, a_j^7$ are trivial, i.e. lines. By the construction of R_{x_j}, the right line of $u_{j,t}$ lies completely to the left of the right line of $w_{j,t}$ for all values of $j = 1, 2, \ldots, n$ and $t = 1, 2, \ldots, m_j$. Furthermore, the right line of each of $\{u_{j,t}, w_{j,t}\}$ lies completely to the left of the right line of each of $\{u_{j,t'}, w_{j,t'}\}$ in R_{x_j}, whenever $t < t'$.

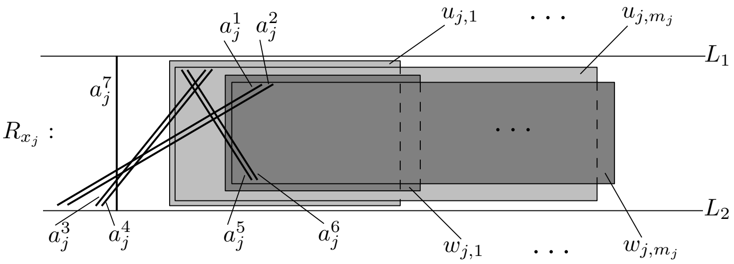


Figure 6 The construction R_{x_j} that corresponds to the variable x_j of the formula ϕ, where $j = 1, 2, \dots, n$.

6.3 The construction the trapezoid graph G_ϕ

We construct now a trapezoid representation R_ϕ of the whole trapezoid graph G_ϕ, by composing the constructions R_{α_i} and R_{x_j} presented in Sections 6.1 and 6.1, as follows. First, we place in R_ϕ the k trapezoid representations R_{α_i}, where $i = 1, 2, \dots, k$, between the lines L_1 and L_2 such that, whenever $i < i'$, every trapezoid of R_{α_i} lies completely to the left of every trapezoid of $R_{\alpha_{i'}}$. Then, we place in R_ϕ the n trapezoid representations R_{x_j}, where $j = 1, 2, \dots, n$, between the lines L_1 and L_2 such that, whenever $j < j'$, the lines of $a_j^1, a_j^2, \dots, a_j^7$ and the left lines of all $u_{j,t}, w_{j,t}$, lie completely to the left of the lines of $a_{j'}^1, a_{j'}^2, \dots, a_{j'}^7$ and the left lines of all $u_{j',t'}, w_{j',t'}$. Moreover, for every $j, j' = 1, 2, \dots, n$, the lines of $a_j^1, a_j^2, \dots, a_j^7$ and the left lines of all $u_{j,t}, w_{j,t}$, lie in R_ϕ completely to the left of the right lines of all $u_{j',t'}, w_{j',t'}$. Thus, note in particular that every $u_{j,t}$ intersects every other $u_{j',t'}$ and every $w_{j',t'}$ in R_ϕ.

Let $j \in \{1, 2, \dots, n\}$ and $t \in \{1, 2, \dots, m_j\}$. Recall that, by the construction of R_{x_j} in Section 6.2, the pair of trapezoids $\{u_{j,t}, w_{j,t}\}$ corresponds to the appearance of the variable x_j in a clause α_i of ϕ, where $i = i_{j,t} \in \{1, 2, \dots, k\}$. That is, either $\ell_{i,p} = x_j$ or $\ell_{i,p} = \overline{x_j}$ for some $p \in \{1, 2, 3\}$, where $\alpha_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$. Then, we place in R_ϕ the right lines of the trapezoids $u_{j,t}$ and $w_{j,t}$ directly before the left line of $v_{i,p}$ (i.e. no line of any other trapezoid intersects with or lies between the right lines of $u_{j,t}$ and $w_{j,t}$ and the left line of $v_{i,p}$).

In order to finalize the construction of R_ϕ, we distinguish now the two cases regarding the literal $\ell_{i,p}$ of the clause α_i, in which the variable x_j appears. If $\ell_{i,p} = x_j$, then we add to R_ϕ six trivial trapezoids (i.e. lines) $\{b_{j,t}^1, b_{j,t}^2, \dots b_{j,t}^6\}$, as it is shown in Figure 7a. On the other hand, if $\ell_{i,p} = \overline{x_j}$, then we add to R_ϕ eight trivial trapezoids (i.e. lines) $\{b_{j,t}^1, b_{j,t}^2, \dots, b_{j,t}^8\}$, as it is shown in Figure 7b. In particular, we place these six (resp. eight) new lines in R_ϕ such that they intersect only the right lines of $u_{j,t}$ and $w_{j,t}$ and the left line of $v_{i,p}$ in R_ϕ. Note that the trapezoid graphs induced by the representations in Figures 7a and 7b are isomorphic to the graphs investigated in Lemmas 10 and 11, respectively. This completes the construction of the trapezoid representation R_ϕ, while G_ϕ is the trapezoid graph induced by R_ϕ.

It is now easy to verify that, by the construction of R_ϕ, all the trapezoids $u_{j,t}$ are upper-left-closed and right-closed in R_ϕ, while all the trapezoids $w_{j,t}$ are lower-right-closed and left-closed in R_ϕ. Furthermore, all the trapezoids $u_{j,t}$ are lower-left-open in R_ϕ and all the trapezoids $w_{j,t}$ are upper-right-open in R_ϕ. Consider now a trapezoid $v_{i,p}$ in R_ϕ. If $v_{i,p}$ corresponds to a positive literal $\ell_{i,p} = x_j$ (for some variable x_j), then $v_{i,p}$ is upper-left-closed and lower-left-open in R_ϕ (cf. Figure 7a). On the other hand, if $v_{i,p}$ corresponds to a negative literal $\ell_{i,p} = \overline{x_j}$, then $v_{i,p}$ is left-closed in R_ϕ (cf. Figure 7b).

We can prove that the formula ϕ is satisfiable if and only if G_ϕ is a triangle graph, cf. [19]. Therefore, since 3SAT is NP-complete, it follows that the recognition of triangle graphs

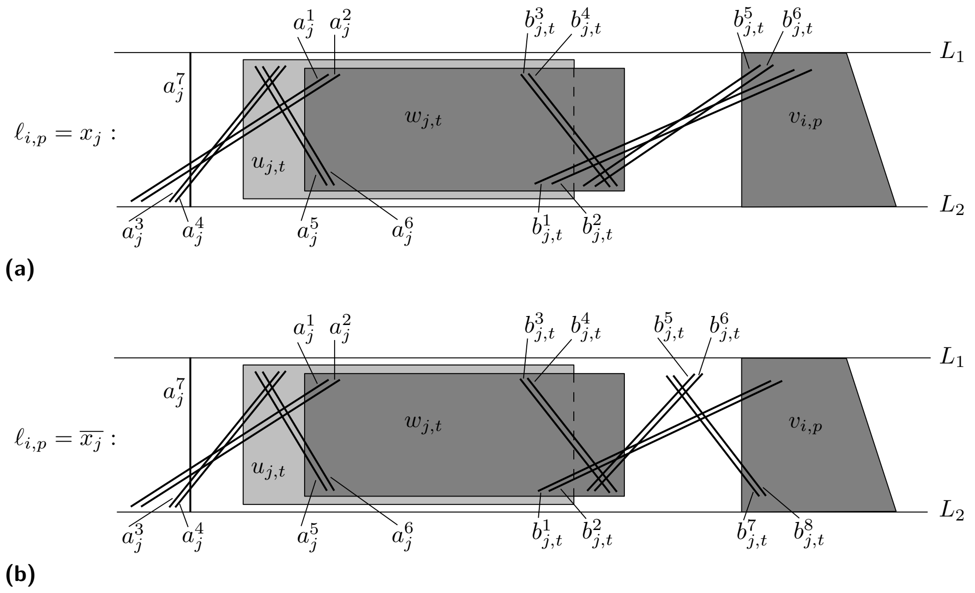


Figure 7 The composition of the trapezoids of R_{x_j} with the trapezoid $v_{i,p}$ of R_{α_i}, in the cases where (a) $\ell_{i,p} = x_j$ and (b) $\ell_{i,p} = \overline{x_j}$.

is NP-hard. Moreover, since the recognition of triangle graphs lies in NP by Observation 1, and since G_ϕ is a trapezoid graph, we can summarize our main result in the next theorem.

▶ **Theorem 14.** *Given a graph G, it is NP-complete to decide whether G is a triangle graph. The problem remains NP-complete even if the given graph G is known to be a trapezoid graph.*

7 Concluding Remarks

In this article we proved that the triangle graph (known also as PI* graph) recognition problem is NP-complete, by providing a reduction from the 3SAT problem, thus answering a longstanding open question. Our reduction implies that this problem remains NP-complete even in the case where the input graph is a trapezoid graph. The recognition of *simple-triangle* graphs [3], as well as the recognition of the related classes of *unit* and *proper tolerance* graphs [1,10] (these are subclasses of bounded tolerance, i.e. parallelogram, graphs [1]), *proper bitolerance* graphs [2,10] (they coincide with *unit bitolerance* graphs [2]), and *multitolerance* graphs [20] (they naturally generalize trapezoid graphs [20,24]) remain interesting open problems for further research.

References

1 Kenneth P. Bogart, Peter C. Fishburn, Garth Isaak, and Larry Langley. Proper and unit tolerance graphs. *Discrete Applied Mathematics*, 60(1-3):99–117, 1995.

2 Kenneth P. Bogart and Garth Isaak. Proper and unit bitolerance orders and graphs. *Discrete Mathematics*, 181(1-3):37–51, 1998.

3 Andreas Brandstädt, Van Bang Le, and Jeremy P. Spinrad. *Graph classes: a survey*. Society for Industrial and Applied Mathematics (SIAM), 1999.

4 Márcia R. Cerioli, Fabiano de S. Oliveira, and Jayme Luiz Szwarcfiter. Linear-interval dimension and PI orders. *Electronic Notes in Discrete Mathematics*, 30:111–116, 2008.

5 Derek G. Corneil and P. A. Kamula. Extensions of permutation and interval graphs. In *Proceedings of the 18th Southeastern Conference on Combinatorics, Graph Theory and Computing*, pages 267–275, 1987.

6 Ido Dagan, Martin Charles Golumbic, and Ron Yair Pinter. Trapezoid graphs and their coloring. *Discrete Applied Mathematics*, 21(1):35–46, 1988.

7 S. Felsner, R. Müller, and L. Wernisch. Trapezoid graphs and generalizations, geometry and algorithms. *Discrete Applied Mathematics*, 74:13–32, 1997.

8 P. C. Fishburn and W. T. Trotter. Split semiorders. *Discrete Mathematics*, 195:111–126, 1999.

9 M. C. Golumbic. *Algorithmic graph theory and perfect graphs (Annals of Discrete Mathematics, Vol. 57).* North-Holland Publishing Co., 2004.

10 M. C. Golumbic and A. N. Trenk. *Tolerance Graphs.* Cambridge studies in advanced mathematics, 2004.

11 Martin Charles Golumbic and Robert E. Jamison. Edge and vertex intersection of paths in a tree. *Discrete Mathematics*, 55(2):151–159, 1985.

12 G. Isaak, K. L. Nyman, and A. N. Trenk. A hierarchy of classes of bounded bitolerance orders. *Ars Combinatoria*, 69, 2003.

13 Ekkehard Köhler. Connected domination and dominating clique in trapezoid graphs. *Discrete Applied Mathematics*, 99(1-3):91–110, 2000.

14 Larry J. Langley. A recognition algorithm for orders of interval dimension two. *Discrete Applied Mathematics*, 60(1-3):257–266, 1995.

15 Yaw-Ling Lin. Triangle graphs and their coloring. In *Proceedings of the International Workshop on Orders, Algorithms, and Applications (ORDAL)*, pages 128–142. Springer-Verlag, 1994.

16 Tze-Heng Ma and Jeremy P. Spinrad. On the 2-chain subgraph cover and related problems. *Journal of Algorithms*, 17(2):251–268, 1994.

17 Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999.

18 T. A. McKee and F. R. McMorris. *Topics in intersection graph theory.* SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 1999.

19 George B. Mertzios. The recognition of triangle graphs. Technical Report CS-2010-17, Department of Computer Science, Technion, Israel Institute of Technology, 2010. http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi/2010/CS/CS-2010-17.

20 George B. Mertzios. An intersection model for multitolerance graphs: Efficient algorithms and hierarchy. In *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2011. To appear.

21 George B. Mertzios and Derek G. Corneil. Vertex splitting and the recognition of trapezoid graphs. Technical Report AIB-2009-16, Department of Computer Science, RWTH Aachen University, September 2009. http://sunsite.informatik.rwth-aachen.de/Publications/AIB/2009/2009-16.pdf.

22 George B. Mertzios, Ignasi Sau, and Shmuel Zaks. The recognition of tolerance and bounded tolerance graphs. In *Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 585–596, 2010.

23 M. Middendorf and F. Pfeiffer. On the complexity of recognizing perfectly orderable graphs. *Discrete Mathematics*, 80(3):327–333, 1990.

24 Andreas Parra. Triangulating multitolerance graphs. *Discrete Applied Mathematics*, 84(1-3):183–197, 1998.

25 Stephen P. Ryan. Trapezoid order classification. *Order*, 15:341–354, 1998.

26 Jeremy P. Spinrad. *Efficient graph representations*, volume 19 of *Fields Institute Monographs.* American Mathematical Society, 2003.

Collapse Operation Increases Expressive Power of Deterministic Higher Order Pushdown Automata*

Paweł Parys

University of Warsaw
ul. Banacha 2, 02-097 Warszawa,Poland
parys@mimuw.edu.pl

Abstract

We show that collapsible deterministic second level pushdown automata can recognize more languages than deterministic second level pushdown automata (without collapse). This implies that there exists a tree generated by a second level recursion scheme which is not generated by any second level safe recursion scheme.

1998 ACM Subject Classification F.4.3 Formal Languages

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.603

1 Introduction

In verification we often approximate an arbitrary program by a program with variables from a finite domain, remembering only a part of information. Then the outcome of some conditions in the program (e.g. in the *if* or *while* statements) cannot be determined, hence they are replaced by a nondeterministic choice (branching). If the program does not use recursion, the set of its possible control flows is a regular language, and the program itself is (in a sense) a deterministic finite automaton recognizing it. If the program contains recursion, we get a deterministic context free language, and from the program one can construct a deterministic pushdown automaton (PDA for short) recognizing this language. In other words, stack can be used to simulate recursion (notice that the same is true for compilers: they convert a recursive program into a program using stack). In verification it is interesting to analyze the possibly infinite tree of all possible control flows of a program. This tree has a decidable MSO theory [4].

A next step is to consider higher order programs, i.e. programs in which procedures can take procedures as parameters. Such programs closely correspond to so-called higher order recursion schemes and to typed λ-terms. They no longer can be simulated by classical PDA. Here higher order PDA come into play. They were originally introduced by Maslov [10]. In automata of level n we have a level n stack of level $n-1$ stacks of ... of level 1 stacks. The idea is that the PDA operates only on the topmost level 1 stack, but additionally it can make a copy of the topmost stack of some level, or can remove the topmost stack of some level. However the correspondence between higher order automata and recursion schemes (programs) is not perfect. Trees recognized (in suitable sense) by a deterministic PDA of level n coincide with higher order recursion schemes of level n with *safety* restriction [7]. See [3, 5] for another characterizations of the same hierarchy. It is important that these trees have decidable MSO theory [7].

To overcome the safety restriction, a new model of pushdown automata were introduced, called collapsible higher order PDA [8, 1]. These automata are allowed to perform an

* Work supported by Polish government grant no. N206 008 32/0810.

additional operation called *collapse* (or *panic* in [8]); it allows to remove all stacks on which a copy of the currently topmost stack symbol is present. These automata correspond to all higher order recursion schemes (not only safe ones) [6], and trees generated by them also have decidable MSO theory [11]. It is also worth to mention that verification of some real life higher order programs can be performed in reasonable time [9].

A question arises if these two hierarchies are possibly the same hierarchy? This is an open problem stated in [7] and repeated in other papers concerning higher order PDA [8, 2, 11, 6]. We give a negative answer to this question, which is our main theorem.

▶ **Theorem 1.** *There exists a language recognized by a collapsible deterministic second level pushdown automaton, which is not recognized by any deterministic second level pushdown automaton without collapse.*

From the equivalences mentioned above we get the following.

▶ **Corollary 2.** *There exists a tree generated by a second level recursion scheme, which is not generated by any safe second level recursion scheme.*

This confirms that the correspondence between higher order recursion schemes and deterministic higher order PDAs is not perfect. The language used in Theorem 1 comes from [7] and from that time was conjectured to be a good candidate.

Related work

One may ask a similar question for nondeterministic automata rather than for deterministic ones. This is an independent problem. The answer is known only for level 2 and is opposite. One can see that for level 2 the collapse operation can be simulated by nondeterminism, hence normal and collapsible nondeterministic level 2 PDA recognize the same languages [2]. However it seems that in context of verification considering deterministic automata is a more natural choice, for the following reasons. First, most problems for nondeterministic PDA are not decidable: even the very basic problem of universality for level 1 PDA is undecidable. Second, we want to verify deterministic programs (possibly with some not deterministic input). A nondeterministic program is something rather strange: it has an oracle which says what to do in order to accept. Normally, when a program is going to make some not deterministic choice, we want to analyze all possibilities, not only these which are leading to some „acceptance” (hence we have branching, not nondeterminism).

2 Definition

A *deterministic second level pushdown automaton* (D2PDA for short) is given by a tuple $(A, \Gamma, \gamma_I, Q, q_I, \delta)$ where A is an input alphabet, Γ is a stack alphabet, γ_I is an initial stack symbol, Q is a set of states, q_I is an initial state, and $\delta\colon Q \times \Gamma \to \mathit{Ops}$ is a transition function. The set Ops contains the following operations: (pop, q), $(\mathsf{push}(\gamma), q)$, (copy, q), $\mathsf{read}_0(t)$, $\mathsf{read}_{acc}(t)$ for each $q \in Q$, $\gamma \in \Gamma$, and $t\colon A \to Q$.

A first level stack is a nonempty sequence of elements of Γ. A second level stack is a nonempty sequence of first level stacks. A configuration of a D2PDA consists of a second level stack, a state from Q, and a head position over the input word. At the beginning on the second level stack there is one first level stack, which contains one γ_I symbol, the state is q_I, and the head is before the first letter of the input word. The automaton always sees only the last (topmost) symbol on the last (topmost) stack. When the current state is q and the last stack symbol is γ, the automaton looks at the transition $\delta(q, \gamma)$ and:

- if $\delta(q, \gamma) = (\mathsf{pop}, q')$, it removes the last symbol from the last first level stack; when the stack becomes empty, it is removed from the second level stack; when the second level stack becomes empty, the automaton fails; the state becomes q';
- if $\delta(q, \gamma) = (\mathsf{push}(\gamma'), q')$, it places symbol γ' on the end of the last first level stack; the state becomes q';
- if $\delta(q, \gamma) = (\mathsf{copy}, q')$, it places a copy of the last first level stack on the end of the second level stack; the state becomes q';
- if $\delta(q, \gamma) = \mathsf{read}_0(t)$, it moves the head to the next letter of the input word; if it is a, the state becomes $t(a)$; if we are on the end of the word, the automaton fails;
- if $\delta(q, \gamma) = \mathsf{read}_{acc}(t)$, it moves the head to the next letter of the input word; if it is a, the state becomes $t(a)$; if we are on the end of the word, the automaton accepts the word.

Notice that none of the stacks is empty; if the last element of a stack is removed, we remove also the whole stack from the second level stack.

Now we are going to define *collapsible D2PDA*. Its first level stacks together with each symbol $\gamma \in \Gamma$ contain a number $n \in \mathbb{N}$ (hence stacks contain pairs (γ, n)). The operation $\mathsf{push}(\gamma)$ places a pair (γ, n) on the end of the last first level stack, where n is the number of first level stacks. We additionally have an operation $(\mathsf{collapse}, q')$ for each $q' \in Q$. When the last element of the last stack is (γ, n) and this operation is performed, we remove all stacks except the first $n-1$ stacks; if $n-1=0$ the automaton fails; the state becomes q'. In other words, $\mathsf{collapse}$ removes all stacks on which a copy of this (γ, n) symbol is present.

An example of a collapsible D2PDA is given in the next section. In the literature one can find some slightly different definitions of a D2PDA and a collapsible D2PDA, but one can see that they are equivalent to ours, through some encodings.

3 The language

Let $A = \{[,], *\}$. A word $w \in \{[,]\}^*$ is called a *prefix of a bracket expression* if in each prefix v of w the number of closing brackets is not greater than the number of opening brackets. A word $w \in \{[,]\}^*$ is called a *bracket expression* if it is a prefix of a bracket expression and the number of opening brackets in w is equal to the number of closing brackets in w. Let PBE and BE be the set of all prefixes of bracket expressions and of all bracket expressions, respectively. For $w \in PBE$ by $open(w)$ we denote the number of [characters in w minus the number of] characters in w (i.e. the number of opened brackets which are not closed). For each $w \in PBE$ we define a number $char(w)$ as $|w| - |v|$ where v is the longest suffix of w, which is a bracket expression. This number is called later a *characteristic* of the word w. We consider the following language over A:

$$U = \{w *^{char(w)+1} : w \in PBE\}.$$

The words $[\,][[\,]****$, $[[[****$, $[\,]*$ are examples of words in U, and $[[[***$ and $[\,]]*$ are examples of words not in U (moreover, no word beginning with $[[\,]$ is in U).

It is known that U can be recognized by a collapsible D2PDA, but for completeness we show it below. The collapsible D2PDA will use three stack symbols: X (used to mark the bottom of stacks), Y (used to count brackets), Z (used to mark the first stack). Initially, the only stack contains one X symbol. The automaton first pushes Z, makes a copy, and pops Z (hence the first stack is marked with Z, the other stacks are used later). Then, for an opening bracket we push Y and we make a copy; for a closing bracket we pop Y and we make a copy. Hence for each bracket we have a stack and on the last stack we have as many

Y symbols as the number of currently open brackets. If for a closing bracket the topmost symbol is X, we fail: it means that the input is not a prefix of a bracket expression.

Finally a star is read. If the topmost symbol is X, we have read a bracket expression, hence we should accept after one star. Otherwise, the topmost Y symbol corresponds to the last opening bracket which is not closed. We do the collapse operation. It leaves the stacks corresponding to the earlier brackets (and the first stack), hence the number of stacks is precisely equal to the characteristic. Now we should read as many stars as we have stacks, and accept (after each star we remove one stack).

4 The proof

In this section we show that U is not recognized by any D2PDA. Assume otherwise: there exists a D2PDA $\mathcal{A}$ recognizing U.

By $\sim$ we denote the Myhill-Nerode relation with respect to U: we have $v \sim w$ if for all u it holds $vu \in U \Leftrightarrow wu \in U$. Notice, in particular, that $open(v) \neq open(w)$ implies $v \not\sim w$.

Our first goal is to eliminate situations in which the number of stacks decreases. As a first step we will eliminate situations in which the number of stacks is first increased and later after a long time decreased to the same value. The intuition is as follows. Consider a run of $\mathcal{A}$ on a word w and a moment when the number of stacks increases from some s to $s+1$. It happens when the head is over some position i of the input word. Let j be the position of the head in the moment when the number of stacks becomes again s (for the first time). Assume that such j exists and that $j-i$ is big. This can happen only if we have a very bad luck. Indeed, consider a word w' get from w by some modification between positions i and j, and assume that in w' the number of stacks also comes down to s at some moment. Notice that in the moment when the number of stacks becomes s, we have the same stacks content for w and for w', the only difference is the state. We have only a fixed number of states and very many nonequivalent modifications of w, which have to give a different state. Hence in most cases either the number of stacks goes down to s very quickly after i, or it stays always above s.

It is formalized using fillings. We say that a function $\sigma\colon PBE \to PBE$ is a *filling*, when

- $\sigma(\epsilon) = \epsilon$, and
- for any $vb \in PBE$ (where $b \in \{[,]\}$), it holds that $\sigma(vb) = \sigma(v)eb$ for some bracket expression e (which may depend on both v and b).

Hence a filling of a word is received by inserting a bracket expression before each letter, but in a deterministic way. A filling is called a *k-filling*, when additionally the length of each inserted bracket expressions e is at most k. We have the following lemma.

▶ **Lemma 3.** *Assume $\mathcal{A}$ is a D2PDA recognizing U. Then there exist constants k, l and a k-filling σ such that if $\mathcal{A}$ reading $\sigma(w)$ for some $w \in PBE$ increases the number of stacks from some s to $s+1$ with the head over a position i, then either the number of stacks is decreased to s for the head over a position $\leq i+l$, or it stays above s for the rest of the run.*

Proof. The constant $k = O(|Q|^2)$ will follow from the proof; we take $l = 2(k+1)$. Our filling σ will satisfy the following additional assumption for any $w \in PBE$:

⋆ Let s_0 be the minimal number of stacks when the head of $\mathcal{A}$ is over one of the last $k+1$ positions of $\sigma(w)$. Then for any $e \in PBE$, the number of stacks never goes below s_0 while $\mathcal{A}$ reads the suffix $[e$ of $\sigma(w)[e$.

We will not define the filling explicitly. Instead, we define it in a non explicit way by induction. We construct the values of filling σ starting from shorter words and going towards longer.

For the empty word we take $\sigma(\epsilon) = \epsilon$; property $\star$ is satisfied ($s_0 = 1$ and we can not have less than one stack), as well as the thesis of the lemma.

Now consider any longer word $vb \in PBE$ (where b is a letter). Let first define some words. For any number n and any function $f\colon \{1, \dots, n\} \to \{0, 1\}$ we define inductively

$$o_f^n = \begin{cases} \epsilon & \text{when } n = 0, \\ o_f^{n-1}[& \text{when } f(n) = 0, \\ o_f^{n-1}[\,][& \text{when } f(n) = 1. \end{cases}$$

Hence o_f^n consists of n opening brackets, and before i-th of them we insert $[\,]$ if $f(i) = 1$. Let s_0 be the minimal number of stacks when the head of $\mathcal{A}$ is over one of the last $k+1$ positions of $\sigma(v)$. Let s_1 be the number of stacks after $\sigma(v)$ is read (precisely, in the moment of the read operation moving the head from the last letter of $\sigma(v)$). Let d be the greatest number ($s_0 \le d \le s_1$) such that for some $f\colon \{1, \dots, |Q| + 1\} \to \{0, 1\}$ the number of stacks never goes below d while $\mathcal{A}$ reads the added suffix of $\sigma(v)[o_f^{|Q|+1}$ (by the *added suffix* we mean the part after $\sigma(v)$). From the $\star$ property it follows that s_0 satisfies this, hence d exists. Fix the particular function f, for which the number of stacks never goes below d while $\mathcal{A}$ reads the added suffix of $\sigma(v)[o_f^{|Q|+1}$.

Consider the functions $f_1, \dots, f_{|Q|+1}$ which differ from f only on one position, namely $f_i(i) \neq f(i)$ and $f_i(j) = f(j)$ for all $j \neq i$. We will show that for at most $|Q|$ of them the number of stacks goes below d while $\mathcal{A}$ reads the added suffix of $\sigma(v)[o_{f_i}^{|Q|+1}e$ for some $e \in PBE$. To see this, for each such f_i fix some e_i (if exists) such that the number of stacks goes below d while $\mathcal{A}$ reads the added suffix of $\sigma(v)[o_{f_i}^{|Q|+1}e_i$. Consider any two such functions f_i and f_j ($i < j$). Let x_i be the prefix of $\sigma(v)[o_{f_i}^{|Q|+1}e_i$ such that the number of stacks decreases to $d - 1$ when the head is over the last letter of x_i. Similarly for j. The key point is that neither x_i nor x_j can be a prefix of $\sigma(v)[o_f^{|Q|+1}$ (i.e. x_i has to contain some letters which are different for f_i than for f), as for this word the number of stacks stays at least d. Assume first that $x_i = \sigma(v)[o_{f_i}^{i-1}[\,]$ (which is possible for $f_i(i) = 1$). But x_j contains at least $\sigma(v)[o_{f_j}^{j-1}$, so $open(x_i) < open(x_j)$, hence $x_i \not\sim x_j$ (recall that $\sim$ is the Myhill-Nerode relation). The other case is that x_i contains at least $\sigma(v)[o_{f_i}^{i}$. When $open(x_i) \neq open(x_j)$, we also have $x_i \not\sim x_j$. When $open(x_i) = open(x_j)$, consider z which closes $open(x_i) - open(\sigma(v)[o_{f_i}^{i})$ brackets. We have $char(x_i z) = |\sigma(v)[o_{f_i}^{i}|$ and $char(x_j z) = |\sigma(v)[o_{f_j}^{i}| = char(x_i z) \pm 2$, hence in this case also $x_i \not\sim x_j$. This means that in the moment when the number of stacks becomes $d - 1$, the state has to be different for i and j (as the stacks content is the same, but the read inputs are not equivalent). As we have only $|Q|$ states, the number of stacks may become $d - 1$ only for at most $|Q|$ functions f_i. Thus there is g (one of $f_1, \dots, f_{|Q|+1}$) such that for each $e \in PBE$, the number of stacks stays at least d while $\mathcal{A}$ reads the added suffix of $\sigma(v)[o_g^{|Q|+1}e$.

Now consider the words (in BE)

$$u_i = \sigma(v)[o_g^{|Q|+1}[^i]^{i+|Q|+2}b$$

for i being a multiple of $|Q| + 3$. We will show that for at most $|Q|$ of them the number of stacks goes below d while $\mathcal{A}$ reads their added suffixes. To see this take any two such words u_i and u_j ($i < j$). Let x_i be the prefix of u_i such that the number of stacks decreases to $d - 1$ when the head is over the last position of x_i; similarly x_j for u_j. We know from the above that the number of stacks can not be decreased to $d-1$ inside $[o_g^{|Q|+1}[^j]^j$ (it is true for $[o_g^{|Q|+1}e$ for any $e \in PBE$, in particular for $e = [^j]^j$), hence $|x_j| \ge |u_j| - |Q| - 2$. However $|u_j| \ge |u_i| + |Q| + 3$ and $|u_i| \ge |x_i|$, which gives $|x_j| > |x_i|$. Thus the characteristics of $x_i[$

and $x_j[$ are different, $x_i \not\sim x_j$. This means that the number of stacks is decreased to $d-1$ in a different state in these two words. Thus only in $|Q|$ words u_i the number of stacks may go below d.

Observe also that there are at most $|Q|$ words u_i in which for some $e \in PBE$ the number of stacks goes below d in the part $[e$ of $u_i[e$, but stays at least d inside the added suffix of u_i. To see this, for each such i fix some $e_i \in PBE$ (if exists) such that the number of stacks goes below d in the part $[e_i$ of $u_i[e_i$. We may assume that this happens when the head is over the last letter of $u_i[e_i$; otherwise the last letter of e_i is redundant and can be cut off. Take any two such words u_i and u_j $(i < j)$. If $\mathit{open}(e_i) \neq \mathit{open}(e_j)$, we have $u_i[e_i \not\sim u_j[e_j$. Otherwise, let z consist of $\mathit{open}(e_i)$ closing brackets; see that $\mathit{char}(u_i[e_iz) = |u_i[|$ and $\mathit{char}(u_j[e_jz) = |u_j[|$. But the lengths of u_i and u_j are different, hence $u_i[e_i \not\sim u_j[e_j$. Thus when the number of stacks is decreased to $d-1$, the state for i and for j has to be different. As we have only $|Q|$ states, there are at most $|Q|$ such words.

From the above two paragraphs it follows that we may choose u_i for $i \leq (2|Q|+1)(|Q|+3)$ such that for each $e \in PBE$ the number of stacks stays at least d while $\mathcal{A}$ reads the added suffix of $u_i[e$ (both inside and outside u_i). As k we take the maximal length of the expression inserted for any such u_i. Observe that this u_i satisfies both the thesis of the lemma and property $\star$. Indeed, whenever the number of stacks decreases from some $s+1$ to s during the added suffix of u_i, then $s \geq d \geq s_0$, hence the number of stacks was increased from s to $s+1$ during the last $l = 2(k+1)$ letters of u_i (and when the decrease is inside $\sigma(v)$, everything is OK from the induction assumption). From the method how d was chosen follows that at some moment while $\mathcal{A}$ reads the added suffix of u_i, the number of stacks is d (even inside the $[o_g^{|Q|+1}$ fragment). On the other hand, for each $e \in PBE$ the number of stacks never goes below d while reading the part $[e$ of $u_i[e$. Thus the $\star$ property is also satisfied. ◀

The next lemma eliminates also all other situations in which the number of stacks is increased from some s to $s+1$ for the head over one position of the word, and then it is decreased from $s+1$ to s over any of the next positions of the word (not only farther than l letters).

▶ **Lemma 4.** *Assume there exists a D2PDA recognizing U. Then there exists a constant k, a k-filling σ and a D2PDA $\mathcal{A}'$ recognizing U such that if $\mathcal{A}'$ reading $\sigma(w)$ for some $w \in PBE$ increases the number of stacks from some s to $s+1$ with the head over a position i, then either the number of stacks is decreased to s for the head over the position i, or it stays above s for the rest of the run.*

Proof. Let $\mathcal{A}$ be a D2PDA recognizing U. The constant k and the k-filling σ are taken from Lemma 3. We have to improve $\mathcal{A}$ such that the stronger property will be satisfied. The automaton $\mathcal{A}'$ remembers a state q of $\mathcal{A}$ and up to l previous letters of the input (where l is the constant from Lemma 3), i.e. a state of $\mathcal{A}'$ contains a state of $\mathcal{A}$ and a sequence of up to l letters (called a *buffer*). We begin with the initial state of $\mathcal{A}$, and no letters in the buffer. When the number of remembered letters is smaller than l, we read the next letter and we append it to our buffer. When the buffer is full (contains l letters), we start executing $\mathcal{A}$. First, we execute $\mathcal{A}$ from the remembered state q until the moment when it reads a letter (we give him the first letter from the buffer). Then, consider also the further run of $\mathcal{A}$, which reads all the next letters of the buffer (until the moment when $\mathcal{A}$ wants to make a read operation when no more letters are in the buffer). We execute the part of this run up to the moment when the number of stacks is minimal (to the last such moment if there are more than one); we describe below how to detect this moment. Denote this minimal number of stacks as s_0. Of course after a letter is read by the run of $\mathcal{A}$ being simulated, we remove it from the buffer.

When $\mathcal{A}'$ sees a star, it executes $\mathcal{A}$ on the letters left in the buffer, and then it simply emulates $\mathcal{A}$ on the rest of the word. Of course $\mathcal{A}$ and $\mathcal{A}'$ accept the same language, as they in fact perform the same operations on a given input word, the only difference is when the read operations are done (in $\mathcal{A}'$ we earlier read more letters and later perform the other operations). Note, in particular, that in the part reading brackets, $\mathcal{A}'$ may always use the read_0 operation, as all words in U have at least one star.

Observe that when such $\mathcal{A}'$ reads a word $\sigma(w)$, the thesis of our lemma is satisfied. Indeed, when the number of stacks is increased from some $s \geq s_0$ to $s+1$, then it decreases back to s before the head is moved (as the head is moved with s_0 stacks). On the other hand, when the number of stacks is increased from some $s < s_0$ to $s+1$, it is done by $\mathcal{A}$ before reading the first letter of the buffer. Later $\mathcal{A}$ does not decrease the number of stacks below s_0 (hence to s) when the head is over any of the next l positions. Thus $\mathcal{A}$ never does this (from the thesis of Lemma 3), hence $\mathcal{A}'$ also.

How to create such $\mathcal{A}'$? The difficulty is that $\mathcal{A}'$ has to find the moment in which the number of stacks is minimal. However it can be done. The part always executed (i.e. up to the first read) is executed in a normal way. Then the rest is executed, but each new stack (created by the copy operation) is marked by the state of $\mathcal{A}$ before the copy operation and by the head position of $\mathcal{A}$ (i.e. how many letters of the buffer were read). More precisely, for the last stack the marking is remembered in the state of $\mathcal{A}'$; for the previous stacks a special stack symbol is put on the top of a stack when the number of stacks is increasing, and is taken from the top of a stack after the number of stacks decreases. Finally, after the whole run reading the buffer is executed, we remove all the stacks with the markings. This gives us s_0 stacks (the minimal number of stacks during the second part of the run). The marking of the last removed stack gives us the new state q of $\mathcal{A}$, and the number of letters which should be removed from the buffer. ◀

In the next lemma we go even further and we eliminate all situations in which the number of stacks is decreased.

▶ **Lemma 5.** *Assume there exists a D2PDA recognizing U. Then there exists a constant k, a k-filling σ and a D2PDA $\mathcal{A}$ recognizing U such that $\mathcal{A}$ reading $\sigma(w)$ for some $w \in PBE$ never decreases the number of stacks.*

Proof. The constant k and the filling σ is taken from Lemma 4 (hence also from Lemma 3). Let $\mathcal{A}'$ be the automaton from Lemma 4; we will improve it, getting an automaton $\mathcal{A}$. We enrich the stack alphabet: together with each stack symbol we keep a function $f\colon Q \to Q \cup \{\mathsf{nr}\}$, where Q is the set of states of $\mathcal{A}'$. The function lying on an i-th place of an s-th stack is defined in the following way. Consider the situation when all stacks after s are removed and all symbols from the s-th stack above the i-th symbol are removed (i.e. the function lies on the topmost place of the last stack). Let start the automaton from a state q. We look at the run until it tries to do a read operation, or until the number of stacks is decreased to $s-1$. When the read operation is first, we assign $f(q) = \mathsf{nr}$. When the decrease is first, and it results in a state p, we assign $f(q) = p$. It is also possible that the run is infinite (it loops in some stupid way), then we also assign $f(q) = \mathsf{nr}$.

The claim is that we can modify the automaton $\mathcal{A}'$ (getting $\mathcal{A}''$) so that it puts on the stack the correct f function together with each symbol. This is because f lying together with some symbol somewhere on a stack depends only on this symbol and on the function

f one place below. In particular it depends only on the contents of the current stack, hence after making a copy of a stack, the functions in the copy stay correct.

Now we make one more modification of $\mathcal{A}''$, getting $\mathcal{A}$. Whenever $\mathcal{A}''$ is going to do the copy operation, we look at the f function of the topmost stack symbol. When $f(q) = \mathsf{nr}$ we really do the copy operation. Otherwise we immediately move to state $f(q)$ without any operation (formally, as each transition has to do some operation, we may for example push something to the stack and then pop it). We may do this, since the automaton $\mathcal{A}''$, after some work with the copy, would also return to the same stack configuration in state $f(q)$. Hence $\mathcal{A}$ accepts the same words as $\mathcal{A}'$.

Observe that the automaton $\mathcal{A}$ never increases and then decreases the number of stacks, without reading any letter in between (as in such situation it makes the „shortcut" described above). When a word $\sigma(w)$ is read, the decrease can not happen also after reading a letter (from Lemma 4). Hence $\mathcal{A}$ never decreases the number of stacks while reading $\sigma(w)$. ◀

The next lemma says that the automaton can know at each moment if the word read already is a prefix of a bracket expression or not. To formalize this, we replace the read_0 operation by two operations: read_0^{PBE} and read_0^{bad}.

▶ **Lemma 6.** *Assume there exists a D2PDA recognizing U. Then there exists a D2PDA $\mathcal{A}$ recognizing U, which instead of read_0 operation uses read_0^{PBE} if the word already read is a prefix of a bracket expression, and read_0^{bad} otherwise. Moreover, there exists a constant k and a k-filling σ such that $\mathcal{A}$ reading $\sigma(w)$ for some $w \in PBE$ never decreases the number of stacks.*

Proof. The constant k and the filling σ is taken from Lemma 5. Let $\mathcal{A}'$ be the automaton from Lemma 5; we will improve it, getting an automaton $\mathcal{A}$. We enrich the input alphabet by a # symbol and we consider the language

$$U' = U \cup \{w\# : w \in PBE\}.$$

We construct first a D2PDA $\mathcal{B}$ recognizing U'. Observe that $w \in PBE$ if $w \in \{[,]\}^*$ and $w*^k \in U$ for some k. Of course $\mathcal{B}$ in its state can remember if the input contained only brackets. Hence, after a # is read, it is enough to check if, after reading some number of stars, the automaton $\mathcal{A}'$ would accept (additionally, when something appears after the # symbol, $\mathcal{B}$ can not accept). It is easy to do so. We make a copy of $\mathcal{A}'$, in which instead of doing a read_0 operation, we assume that a star was read. When $\mathcal{A}'$ does read_{acc}, we accept our word.

An automaton $\mathcal{C}$ (also recognizing U') is constructed using a trick like in the previous lemma. Together with each stack symbol we remember a function $f \colon Q \to Q \cup \{\mathsf{acc}, \mathsf{na}\}$, where Q is the set of states of $\mathcal{B}$. It is defined in the same way as in the proof of the previous lemma, but it distinguishes an accepting and a non accepting read operation: when a run from q leads to a read_{acc} operation, we assign $f(q) = \mathsf{acc}$, and when it leads to read_0 (or the run is infinite), we assign $f(q) = \mathsf{na}$. The automaton can put on the stack the correct function together with each symbol. (One may ask if it is possible that $f(q) \in Q$, i.e. that the automaton decreases the number of stacks. It is possible, because it does not decrease the number of stacks only while reading the filling; here we can read arbitrary words, in particular containing $*$ or # symbols.)

Moreover on the top of each stack except the last we keep a function $g \colon Q \to \{\mathsf{acc}, \mathsf{na}\}$; for the last stack the function is kept in the state of $\mathcal{C}$. The function for an s-th stack is defined in the following way: Assume that there are only the first $s - 1$ stacks; start a run of $\mathcal{B}$ from a state q and continue it to the first read operation. If it is the read_{acc} operation, we take $g(q) = \mathsf{acc}$, otherwise $g(q) = \mathsf{na}$. Notice that g for an s-th stack depends only on g for the $s-1$-th stack (which „describes” first $s-2$ stacks) and on f on the top of the $s-1$-th stack (which „describes” the $s-1$-th stack). Hence g can be computed whenever a copy operation is done.

Finally we construct $\mathcal{A}$. It works like $\mathcal{C}$, but when a read_0 operation is going to be done, we look at f at the current character and g for the current stack. Assume reading a $\#$ character would end in a state q. If $f(q) = \mathsf{acc}$ we make the read_0^{PBE} operation, if $f(q) = \mathsf{na}$ we make the read_0^{bad} operation. Otherwise $f(q)$ is a state; if $g(f(q)) = \mathsf{acc}$ we make the read_0^{PBE} operation, if $g(f(q)) = \mathsf{na}$ we make the read_0^{bad} operation. Note that $\mathcal{A}$ still recognizes U'. Hence when the input alphabet is limited to $\{[,], *\}$, it recognizes U. Moreover, it uses the operation read_0^{PBE} after a word w, when $w\# \in U'$, hence when w is a prefix of a bracket expression. ◀

For the rest of the proof fix the automaton $\mathcal{A}$, the constant k and the k-filling σ, which are the result of Lemma 6.

For any number $n \geq 1$, let

$$w_n = \underbrace{[^{n+1}]^n[^{n+1}]^n \dots [^{n+1}]^n}_{|Q|+1 \text{ times}}.$$

We will see that after reading a word $\sigma(w_n)$, the number of symbols on the last stack has to be small.

▶ **Lemma 7.** *There exists a constant H such that for any $n \geq 1$ after reading the word $\sigma(w_n)$ the number of symbols on the last stack of $\mathcal{A}$ is not greater than H.*

Proof. For each prefix u of the word $\sigma(w_n)$ we define a block number: u is in the first block if u is a prefix of $\sigma([^{n+1})$, in the second block if u is a prefix of $\sigma([^{n+1}]^n)$ but not of $\sigma([^{n+1})$, in the third block if u is a prefix of $\sigma([^{n+1}]^n[^{n+1})$ but not of $\sigma([^{n+1}]^n)$, etc. Observe the following property $\star\star$. Consider two prefixes u_1 and u_2 of $\sigma(w_n)$ in the same block such that $|u_2| \geq |u_1| + (a+2k)(k+1)$ for some $a \geq 0$. We have

$$\begin{aligned} &open(u_2) \geq open(u_1) + a && \text{if the block number is odd,} \\ &open(u_2) \leq open(u_1) - a && \text{if the block number is even.} \end{aligned}$$

Indeed, assume the block number is odd. Consider the word $u_1^{-1}u_2$ (the suffix of u_2 which is after u_1). At the beginning it contains a suffix of a bracket expression (up to k letters), then opening brackets (coming from w_n) alternating with short (up to k letters) bracket expressions, and finally a prefix of a bracket expression (up to k letters). There are at least $a+2k$ opening brackets coming from w_n (as $|u_2| \geq |u_1| + (a+2k)(k+1)$). In the bracket expressions the number of opening and closing brackets is the same. In the initial fragment the balance is violated by at most k; the same for the final fragment.[1] Thus $open(u_2) \geq open(u_1) + a$. For even block number (having closing brackets) we get the opposite inequality.

Now come to a proof of the lemma. It is important that the automaton never decreases the number of stacks (thesis of Lemma 6). Hence, as long as it reads brackets, it can access only symbols on the last stack. Consider the run reading some $\sigma(w_n)$; assume its

[1] In fact, only the prefix or the suffix mattes, not both of them, so we could replace $a+2k$ by $a+k$.

configurations are numbered from 1 to some l, and in the last configuration the number of symbols on the last stack is h. From the last configuration the automaton tries to do the read_0^{PBE} operation. For each i ($1 \leq i \leq h$) let $p(i) - 1$ denote the number of the last configuration in the run such that the number of symbols on the last stack is smaller than i. Hence in the operation leading to configuration $p(i)$ the i-th symbol is pushed on the last stack and later it is never poped. To each i ($1 \leq i \leq h$) we assign a triple (x, q, γ), where $1 \leq x \leq 2(|Q| + 1)$, $q \in Q$, $\gamma \in \Gamma$. Here x is the block number of the prefix already read in configuration $p(i)$, q is the state in configuration $p(i)$, and γ is the stack symbol on position i on the last stack (in all moments between $p(i)$ and l).

There is a constant H (depending on k and $|Q|$) such that whenever $h > H$, some triple (x, q, γ) has to repeat at least $(2k + |Q| + 2)(k + 1) + 1$ times. Assume first that x in this triple is even (i.e. it corresponds to a block of closing brackets). Take any $c = (2k+1)(k+1)$ numbers $i_1 < i_2 < \cdots < i_c$ to which this triple is assigned. For each j, the run after $p(i_j)$ has no access to the symbols below i_j on the last stack (as well as to the symbols on the earlier stacks). Thus it depends only on the i_j-th stack symbol, the state, and the input word. Notice that the run between $p(i_j)$ and $p(i_{j+1})$ does at least one read operation, as otherwise the fragment between $p(i_j)$ and $p(i_{j+1})$ would repeat forever (the automaton is deterministic). Let r be the number of read operations in the run between $p(i_1)$ and $p(i_c)$. We have $r \geq c - 1$. From $\star\star$ it follows that the part of the input returned by these r read operations contains more closing brackets than opening brackets. Let repeat $|Q| + 2$ more times the fragment of the run from $p(i_1)$ to $p(i_c)$ (precisely, we repeat the operations done in this fragment, together with the part of the input returned by the read operations, and we leave the operations done later). We get a correct run on a new word, in particular after the last configuration it also does the read_0^{PBE} operation. But the new input word is not a prefix of a bracket expression, as it has too many closing brackets. This contradicts with the assumption that our automaton satisfies the thesis of Lemma 6, i.e. that it should end now doing the read_0^{bad} operation.

The argument is similar for odd x, but we have to consider $c = (2k + |Q| + 2)(k + 1) + 1$ numbers $i_1 < \cdots < i_c$ to which the repeating triple (x, q, γ) is assigned. As previously, there is at least one read operation between $p(i_j)$ and $p(i_{j+1})$ for each j. Thus the number r of read operations between $p(i_1)$ and $p(i_c)$ is at least $c - 1$. This time we remove the fragment of the run from $p(i_1)$ to $p(i_c)$. From $\star\star$, the part of the input read between $p(i_1)$ and $p(i_c)$ contained at least $|Q| + 2$ more opening brackets than closing brackets. We get the same contradiction as previously, as the new word is not a prefix of a bracket expression. ◀

For any $n \geq 1$, $0 \leq c \leq |Q|$ we will define a number $d(n, c)$. Assume that after reading $\sigma(w_n)$ there are s stacks. Now see what happens when we read the word $\sigma(w_n]^c)*^\omega$, where $*^\omega$ means that we give infinitely many stars to the automaton and we look at the infinite run. We look for the first of the two situations:

1. the automaton accepts (i.e. makes a read_{acc} operation), or
2. the number of stacks goes below s.

Note that for sure the automaton accepts after some number of stars (but possibly the second situations appears earlier). Note also that none of these situations can appear before we start reading the stars: during reading $\sigma(v)$ for any $v \in PBE$ the number of stacks does not decrease, and no word without stars can be accepted. Let $d(n, c)$ be the number of stars after which the earlier of these two situations appears.

Observe that $d(n, c)$ depends only on the content of the last stack (stack s) after reading $\sigma(w_n)$, on the state in this moment, and on the suffix of the filling $\sigma(w_n]^c)$ which appears after $\sigma(w_n)$. This is because the run reading $\sigma(w_n]^c)*^\omega$ never accesses stacks below s, until some of the two interesting situations appear. Hence there are only finitely many possibilities: The number of the suffixes is finite, as their length is bounded by $|Q|(k+1)$. Thanks to Lemma 7 after reading $\sigma(w_n)$ stack s has height not greater than H, so the number of its different contents is finite. Thus there is a common upper bound D for all $d(n,c)$.

Let now fix $n = D$. Let s be the number of stacks after reading $\sigma(w_n)$. We define a partial function $a\colon Q \to \mathbb{N}$. Let us remove the stack s and start the automaton from a state q on the input word $*^\omega$. If in this infinite run $\mathcal{A}$ makes the read_{acc} operation only once, then let $a(q)$ denote the number of stars after which this happens. In the other cases ($\mathcal{A}$ never accepts or accepts multiple times) $a(q)$ is undefined. Let $u_c = \sigma(w_n]^c)$ for $0 \leq c \leq |Q|$. Consider the run on some of the words $u_c{*}^{char(u_c)+1}$. Note that $char(u_c) \geq 2D+1 > D$ (we count at least the length of prefix $\sigma([^{n+1}]^n)$), hence after reading $d(n,c) \leq D$ stars $\mathcal{A}$ can not accept. Thus the number of stacks becomes $s-1$. The rest of the run depends only on the state q in this moment (as the content of the first $s-1$ stacks is the same for each c); the read_{acc} operation will appear after $a(q)$ more stars (in particular $a(q)$ is defined for this q). Hence $char(u_c) - D \leq a(q) \leq char(u_c)$. As there are only $|Q|$ states, and $|Q|+1$ values of c, some state q has to be used for two values of c, say c_1 and c_2 ($c_1 < c_2$). Note that $char(u_{c_1}) \geq char(u_{c_2}) + 2D + 1 > char(u_{c_2}) + D$ as to $char(u_{c_1})$ we count at least two blocks of brackets more than to $char(u_{c_2})$. This is a contradiction, as

$$a(q) \leq char(u_{c_2}) < char(u_{c_1}) - D \leq a(q).$$

5 Future work

The following question remains open: is there a language recognized by a collapsible deterministic higher order pushdown automaton which is not recognized by any deterministic higher order pushdown automaton without collapse *of any level*? It is possible that the language U from Theorem 1 has this property.

References

1 Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong. The monadic second order theory of trees given by arbitrary level-two recursion schemes is decidable. In *TLCA*, pages 39–54, 2005.

2 Klaus Aehlig, Jolie G. de Miranda, and C.-H. Luke Ong. Safety is not a restriction at level 2 for string languages. In *FoSSaCS*, pages 490–504, 2005.

3 Didier Caucal. On infinite terms having a decidable monadic theory. In *MFCS*, pages 165–176, 2002.

4 Bruno Courcelle. The monadic second-order logic of graphs ix: machines and their behaviours. *Theor. Comput. Sci.*, 151(1):125–162, 1995.

5 Bruno Courcelle and Teodor Knapik. The evaluation of first-order substitution is monadic second-order compatible. *Theor. Comput. Sci.*, 281(1-2):177–206, 2002.

6 M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS '08*, pages 452–461, Washington, DC, USA, 2008. IEEE Computer Society.

7 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS '02*, pages 205–222, London, UK, 2002. Springer-Verlag.

8 Teodor Knapik, Damian Niwinski, Pawel Urzyczyn, and Igor Walukiewicz. Unsafe grammars and panic automata. In *ICALP*, pages 1450–1461, 2005.

9 Naoki Kobayashi. Model-checking higher-order functions. In *PPDP '09: Proceedings of the 11th ACM SIGPLAN conference on Principles and practice of declarative programming*, pages 25–36, New York, NY, USA, 2009. ACM.

10 A. N. Maslov. The hierarchy of indexed languages of an arbitrary level. *Soviet Math. Dokl.*, 15:1170–1174, 1974.

11 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS '06*, pages 81–90, Washington, DC, USA, 2006. IEEE Computer Society.

Temporal Synthesis for Bounded Systems and Environments*

Orna Kupferman[1], Yoad Lustig[2], Moshe Y. Vardi[2], and Mihalis Yannakakis[3]

1 **Hebrew University**
2 **Rice University**
3 **Columbia University**

Abstract

Temporal synthesis is the automated construction of a system from its temporal specification. It is by now realized that requiring the synthesized system to satisfy the specifications against all possible environments may be too demanding, and, dually, allowing all systems may be not demanding enough. In this work we study *bounded temporal synthesis*, in which bounds on the sizes of the state space of the system and the environment are additional parameters to the synthesis problem. This study is motivated by the fact that such bounds may indeed change the answer to the synthesis problem, as well as the theoretical and computational aspects of the synthesis problem. In particular, a finer analysis of synthesis, which takes system and environment sizes into account, yields deeper insight into the quantificational structure of the synthesis problem and the relationship between strong synthesis – there exists a system such that for all environments, the specification holds, and weak synthesis – for all environments there exists a system such that the specification holds.

We first show that unlike the unbounded setting, where determinacy of regular games implies that strong and weak synthesis coincide, these notions do not coincide in the bounded setting. We then turn to study the complexity of deciding strong and weak synthesis. We show that bounding the size of the system or both the system and the environment, turns the synthesis problem into a search problem, and one cannot expect to do better than brute-force search. In particular, the synthesis problem for bounded systems and environment is Σ_2^P-complete (in terms of the bounds, for a specification given by a deterministic automaton). We also show that while bounding the environment may lead to the synthesis of specifications that are otherwise unrealizable, such relaxation of the problem comes at a high price from a complexity-theoretic point of view.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.615

1 Introduction

Temporal synthesis is the automated construction of a system from its temporal specification. The basic idea is simple and appealing: instead of developing a system and verifying that it satisfies its specification, we would like to have an automated procedure that, given a specification, constructs a system that is correct by construction. The first formulation of synthesis goes back to Church [4]; the modern approach was initiated by Pnueli and Rosner, who introduced LTL (linear temporal logic) synthesis [22]. The *LTL synthesis problem* receives as input a specification given by means of an LTL formula and outputs a reactive system modeled by a finite-state transducer satisfying the given specification — if such exists.

In the specification to the system, it is important to distinguish between output signals, controlled by the system, and input signals, controlled by the environment. A system should satisfy its specification against all possible environments. Therefore, the quantification structure on input and output

* Supported in part by NSF grants CCF-0613889, CCF-0728882, CCF-0728736, CCF-1017955, and CNS 1049862, by BSF grant 9800096, and by gift from Intel. Part of this work was done while Moshe Y. Vardi was on sabbatical at the Hebrew University in Jerusalem, supported by a Forchheimer Visiting Professorship.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 615–626
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

signals is different. Output signals are existentially quantified while input signals are universally quantified [22]. It is by now realized that requiring the synthesized system to satisfy the specification against all possible environments is often too demanding. Dually, allowing all possible systems is perhaps not demanding enough. This issue is traditionally approached by adding assumptions on the system and/or the environment, which are modeled as part of the specification (c.f., [3]). [1]

In this work we study *bounded temporal synthesis*, in which assumptions on the system and its environment are given by means of bounds on the sizes of their state space. Thus, in addition to a specification ψ, the input to the bounded synthesis problem contains two parameters $n, m \geq 1$, and a specification ψ is (n,m)*-realizable*, if there is a transducer $\mathcal{T}$ with n states such that for all transducers $\mathcal{T}'$ with m states, the computation $\mathcal{T} \| T'$ – generated by the interaction of $\mathcal{T}$ with $\mathcal{T}'$, satisfies ψ. Note that traditional synthesis corresponds to the case $n = m = \infty$. [2] Also note that by setting only one of n or m to ∞, we can consider a setting in which only one of the components is bounded. In particular, [25] studies the setting in which only the system is bounded. We note that the need to bound the environment is of interest in several other paradigms in computer science. For example, in cryptography, one studies the security of a given cryptosystem with respect to attackers with bounded (typically polynomial) computational power [18], and in the analysis of on-line algorithms one sometimes care for the competitive ratio of a given on-line algorithm with respect to requests issued by a bounded adversary [1]. Even closer to the work here is the study of bounded rationality in games, where bounds are placed on the power of the players. In particular, having the players be automata with a bounded number of states is a natural way of doing this. As shown in [20], such bounds affect the kind of equilibria one gets, and gives in fact a way of getting around some of the problematic cases of equilibria, (e.g., in the Prisoner's Dilemma [24]).

It is not hard to see that bounding the size of the system or its environment may indeed change the answer to the synthesis problem. Clearly, already in a setting with no interaction, bounding the size of a system may prevent it from satisfying some specifications. In the presence of interaction, bounding the size of the environment both restricts the possible behaviors of the environment and enables the system to "learn" the environment. For example, knowing that the environment has a single state implies that the input to the system is fixed, thus a specification like "if p holds in the present, then p holds always", for an input signal p, is realizable against environments with a single state, while it is clearly not realizable in general.

Traditional temporal synthesis is *determined*, in the sense that for every specification ψ, either there is a system that realizes ψ, or there is an environment that realizes $\neg\psi$. Note that not having a system that realizes ψ only means that for every system $\mathcal{T}$, there is an environment $\mathcal{T}'$ such that the computation $\mathcal{T} \| \mathcal{T}'$ does not satisfy ψ. This by itself does not imply that there is an environment $\mathcal{T}'$ such that for all systems $\mathcal{T}$, the computation $\mathcal{T} \| \mathcal{T}'$ satisfies $\neg\psi$. However, by determinacy of Borel games [17], we know that the lack of $\mathcal{T}$ that realizes ψ does imply the existence of $\mathcal{T}'$ that realizes $\neg\psi$. We show that determinacy no longer holds in the bounded setting. In particular, for every $k \geq 1$ there is a specification ψ_k such that ψ_k cannot be realized against environments of size k, nor can $\neg\psi_k$ be realized by an environment of size k.

The observation about determinacy, which uses the theory of *checking sequences* for transducers [16], yields deeper insight into the quantificational structure of the synthesis problem and the relationship between two possible definitions of synthesis in the bounded setting: *strong synthesis*, where there is a system $\mathcal{T}$ such that for all environments $\mathcal{T}'$, the computation $\mathcal{T} \| \mathcal{T}'$ satisfies the

[1] A different, more conceptual, way to restrict the range of environments with respect to which the system has to satisfy the specification is to assume that the environment has objectives of its own, and is therefore *rational*, rather than hostile [9].

[2] In fact, by the small model property, already to the case n and m are doubly exponential in the length of $|\psi|$ [7, 22].

specification, and *weak synthesis*, where for all environments $\mathcal{T}'$ there exists a system $\mathcal{T}$ such that the computation $\mathcal{T}\|\mathcal{T}'$ satisfies the specification. In contrast, in the unbounded setting, strong and weak synthesis coincide.

We study the complexity of strong and weak synthesis. Recall that LTL synthesis is 2EXPTIME-complete [22]. This high complexity, along with technical challenges in the implementation of synthesis algorithms [12, 30], have led researchers to develop synthesis algorithms for fragments of LTL or alternative specification formalisms [15, 21]. A different approach for coping with the complexity of LTL synthesis, tightly related to our work here, is to restrict attention to systems of a bounded size [25]. As argued in [25], bounding the size of the system enables a reduction of the synthesis problem to the SAT problem, and also leads to the decidability of synthesis of distributed systems. For the bounded setting, researchers were also able to come up with a symbolic implementation [6, 8]. [3]

Recall that the bounded synthesis problem has three parameters: ψ, n, and m. We would like to study the complexity in terms of each of n and m. One standard way to analyze the complexity in terms of one parameter is to fix the other parameters. This standard way does not work in our setting, as fixing ψ implies fixing also n and m. Indeed, by [7, 22], if ψ is realizable, then it is also realizable by a transducer with doubly-exponentially many states. Also, by determinacy, ψ is not realizable if the environment can realize $\neg\psi$ by a transducer that is doubly-exponential in the length of ψ. Accordingly, we have to neutralize the dominance of ψ in the complexity analysis in a different way. We do so by assuming that the temporal specification is given, instead of as an LTL formula, as a deterministic Büchi automaton. For such specifications, the unbounded realizability problem amounts to checking the nonemptiness of a deterministic Büchi automaton, and can therefore be solved in quadratic time [28]. In Section 4, we justify the choice of deterministic Büchi automata further.

We first show that bounding the size of the system enables an easy reduction from the synthesis problem to the model-checking problem. At the same time, one cannot expect to do better than brute-force search in synthesizing bounded-size systems. Formally, we show that deciding whether a specification $\mathcal{A}$ givens by means of a deterministic Büchi automaton is realizable by a system with n states is NP-complete, for n given in unary. The proof of NP-hardness is technically easy and is similar to known NP-hardness proofs in the context of formal methods [5, 13]. Still, it justifies the reduction to SAT given in [25] without a lower bound, and it sets the stage to the much more challenging lower bound, for the case both the system and the environment are bounded: we show that deciding whether a specification $\mathcal{A}$ is realizable by a system with n states against all environment with m states is Σ_2^P-complete, for n and m given in unary. Thus, brute-force search is the best we can do also here, and one cannot expect to do better than model checking the interaction of all systems with n states with all environments with m states.

We also show that while bounding the environment may indeed lead to the synthesis of specifications that are otherwise unrealizable, such relaxation of the problem comes at a high price from a complexity-theoretic point of view. As pointed above, synthesis with respect to specifications given by deterministic Büchi automata is quadratic. Adding a bound on the size of the environment seems to add a cost that is doubly exponential in that size. In fact, we show that even model checking against bounded environments is apparently harder than standard model checking. Finally, we formalize the intuition of the system being able to learn the bounded environment by showing that for absolute liveness properties, which are insensitive to additions of prefixes [26], weak and strong realizability

[3] As mentioned above, [25] also studies synthesis of bounded systems. The contributions here and in [25] are, however, different. In [25], the focus is on practical algorithm for the setting of a bounded system. Here, the focus is on the theoretical aspects of the problem and its complexity, and rather than studying bounded systems, we consider bounds on the system, the environment, and both.

coincide, and the complexity of the problem is only exponential in the bound for the environment.

Due to the lack of space, some proofs are omitted, and can be found in the full version in the authors' home pages.

2 Preliminaries

Consider finite sets I and O of input and output signals, respectively. We model finite-state reactive systems with inputs in I and outputs in O by *transducers* (I/O-transducers, when I and O are not clear from the context). A transducer is a finite graph with a designated start state, where the edges are labeled by letters in 2^I and the states are labeled by letters in 2^O. Formally, a transducer is a tuple $\mathcal{T} = \langle I, O, S, s_{in}, \eta, L\rangle$, where I and O are the sets of input and output signals, S is a finite set of states, $s_{in} \in S$ is an initial state, $\eta : S \times 2^I \to S$ is a deterministic transition function, and $L : S \to 2^O$ is a labeling function. We extend η to words in $(2^I)^*$ in the straightforward way. Thus, $\eta : (2^I)^* \to S$ is such that $\eta(\varepsilon) = s_{in}$, and for $x \in (2^I)^*$ and $i \in 2^I$, we have $\eta(x \cdot i) = \eta(\eta(x), i)$. Each transducer $\mathcal{T}$ induces a *strategy* $f_\mathcal{T} : (2^I)^* \to 2^O$ where for all $w \in (2^I)^*$, we have $f_\mathcal{T}(w) = \tau(L(w))$. Thus, $f_\mathcal{T}(w)$ is the letter that $\mathcal{T}$ outputs after reading the sequence w of input letters. A transducer with at most k states is referred to as a *k-transducer*.

Consider an infinite sequence $w = i_0, i_1, i_2, i_3, \ldots \in (2^I)^\omega$ of input letters. The computation of $\mathcal{T}$ on w, denoted $\mathcal{T}(w)$, is $\rho = (o_0 \cup i_0), (o_1 \cup i_1), (o_2 \cup i_2), \ldots \in (2^{I\cup O})^\omega$ such that for all $j \geq 0$, we have $o_j = f_\mathcal{T}(i_0 \cdot i_1 \cdots i_{j-1})$. Note that, in particular, $o_0 = f_\mathcal{T}(\epsilon)$. Thus, the mode of interaction we assume is that the transducer initiates the interaction with the environment by outputting $f_\mathcal{T}(\epsilon)$, the environment then responds with i_0 (making $f_\mathcal{T}(\epsilon) \cup i_0$ the set of signals that are valid in the first time unit), then the transducer responds with $f_\mathcal{T}(i_0)$, the environment with i_1, and so on. In order to emphasize the fact that the transducer moves first, we sometimes refer to ρ as a 1*-computation* of $\mathcal{T}$. Also, we sometimes refer to ρ as w/y, for $y = o_0, o_1, o_2 \ldots$.

One could also consider a dual type of interaction, in which the environment moves first. Then, a 2*-computation* of $\mathcal{T}$ is $\rho = (o_0 \cup i_0), (o_1 \cup i_1), (o_2 \cup i_2), \ldots$ where for all $j \geq 0$, we have $o_j = f_\mathcal{T}(i_0 \cdot i_1 \cdots i_j)$. In particular, $o_0 = f_\mathcal{T}(i_0)$. Note that when two transducers interact with each other, one of them initiates the interaction and moves first, thus its computations are 1-computations, and the second moves second, and its computations are 2-computations. We say that $\rho \in (2^{I\cup O})^\omega$ is a computation of $\mathcal{T}$ if there is $w \in (2^I)^\omega$ such that $\rho = \mathcal{T}(w)$. Finally, we sometimes refer also to finite sequences of input letters and the finite computations of $\mathcal{T}$ on them.

We specify on-going behaviors of I/O-transducers by means of LTL formulas over the set $I \cup O$ of atomic propositions, or automata on infinite words over the alphabet $2^{I\cup O}$. For a specification ψ over $I \cup O$ and a "who moves first" flag $b \in \{1, 2\}$, we use $real_{I,O,b}(\psi)$ to indicate that there is an I/O transducer $\mathcal{T}$ such that all the b-computations of $\mathcal{T}$ satisfy ψ. Given a specification ψ over the sets I and O of input and output signals, the *realizability problem* for ψ is to decide whether $real_{I,O,1}(\psi)$ [22]. For $b \in \{1, 2\}$ let $\tilde{b}$ dualize b (that is, $\tilde{b} = 3 - b$). By determinacy of Borel games [17], we have the following (see also [10]).

▶ **Theorem 1.** *For every specification ψ, precisely one of $real_{I,O,b}(\psi)$ or $real_{O,I,\tilde{b}}(\neg\psi)$ holds.*

3 Strong and Weak realizability

The traditional definition of realizability requires ψ to be satisfied in all the computations of $\mathcal{T}$, ignoring the ability of the environment to feasibly generate the sequences of input letters that induce these computations. In this work, we are interested in realizability with components of a bounded size. In order to define this setting, let us first formalize the interaction between two transducers. For an I/O-transducer $\mathcal{T}$ that induces a strategy $f_\mathcal{T} : (2^I)^* \to 2^O$ and an O/I-transducer $\mathcal{T}'$ that induces a strategy $f_{\mathcal{T}'} : (2^O)^* \to 2^I$, the single 1-computation of $\mathcal{T}\|\mathcal{T}'$ is $\rho = (o_0 \cup i_0), (o_1 \cup i_1), (o_2 \cup i_2), \ldots$ where for all $j \geq 0$, we have $o_j = f_{\mathcal{T}}(i_0 \cdot i_1 \cdots i_{j-1})$ and $i_j = f_{\mathcal{T}'}(o_0 \cdot o_1 \cdots o_j)$. The single 2-computation of $\mathcal{T}\|\mathcal{T}'$ is $\rho = (o_0 \cup i_0), (o_1 \cup i_1), (o_2 \cup i_2), \ldots$ where for all $j \geq 0$, we have $i_j = f_{\mathcal{T}'}(o_0 \cdot o_1 \cdots o_{j-1})$ and $o_j = f_{\mathcal{T}}(i_0 \cdot i_1 \cdots i_j)$.

Recall that a specification ψ is realizable if there is an I/O-transducer $\mathcal{T}$ such that all the 1-computations of $\mathcal{T}$ satisfy ψ. It is known that if a transducer $\mathcal{T}$ does have a computation that violates ψ, then $\mathcal{T}$ also has a computation that violates ψ and is induced by an input sequence that is generated by a transducer [2]. Accordingly, ψ is realizable iff there is an I/O-transducer $\mathcal{T}$ such that for all O/I transducers $\mathcal{T}'$, the 1-computation of $\mathcal{T}\|\mathcal{T}'$ satisfies ψ. In Definition 2 below, we call this "strong realizability" and introduce also a weaker type of realizability.

▶ **Definition 2.** [*Strong and Weak Realizability*] Consider a specification ψ over $I \cup O$, a first-move flag $b \in \{1, 2\}$, and $m, n \in \mathcal{N} \cup \{\infty\}$.

- We say that ψ is *strongly* (I, O, b)*-realizable with respect to systems with* n *states and environments with* m *states*, denoted $s\_real_{I,O,b}(\psi, n, m)$, if there is an I/O-transducer $\mathcal{T}$ with at most n states such that for every O/I-transducer $\mathcal{T}'$ with at most m states, the b-computation of $\mathcal{T}\|\mathcal{T}'$ satisfies ψ.
- We say that ψ is *weakly* (I, O, b)*-realizable with respect to systems with* n *states and environments with* m *states*, denoted $w\_real_{I,O,b}(\psi, n, m)$, if for every O/I-transducer $\mathcal{T}'$ with at most m states, there is an I/O-transducer $\mathcal{T}$ with at most n states such that the b-computation of $\mathcal{T}\|\mathcal{T}'$ satisfies ψ.

Strong realizability means that the system can defeat all environment's strategies under consideration. Weak realizability means that the environment does not have a strategy to defeat all the strategies of the system. When strong realizability is impossible, a designer may settle for a weak one. Also, weak realizability of $\neg\psi$ by the environment explains why ψ is not realizable by the system. Formally, we have the following.

▶ **Lemma 3.** *$w\_real_{I,O,b}(\psi, n, m)$ iff not $s\_real_{O,I,\tilde{b}}(\neg\psi, m, n)$.*

As discussed above, $real_{I,O,b}(\psi)$ iff $s\_real_{I,O,b}(\psi, \infty, \infty)$. In fact, as we state below, weak and strong realizability coincide in the unbounded setting.

▶ **Theorem 4.** *For every specification ψ, we have that $w\_real_{I,O,b}(\psi, \infty, \infty)$ iff $s\_real_{I,O,b}(\psi, \infty, \infty)$.*

Proof. By definition, $s\_real_{I,O,b}(\psi, \infty, \infty)$ implies $w\_real_{I,O,b}(\psi, \infty, \infty)$. For the other direction, assume that $s\_real_{I,O,b}(\psi, \infty, \infty)$ does not hold. Then, by Theorem 1, we have that $s\_real_{O,I,\tilde{b}}(\neg\psi, \infty, \infty)$. Thus, by Lemma 3, we have that $w\_real_{I,O,b}(\psi, \infty, \infty)$ does not hold, and we are done. ◀

▶ **Example 5.** Let $I = \{p\}$, $O = \{q\}$, and $\psi = G(q \leftrightarrow Xp)$. Note that the specification requires the next value of the input signal p to depend on the current value of the output signal q. Since the system has no control on the input signals, we have that not $s\_real_{I,O,1}(\psi, \infty, \infty)$, and in fact even not $s\_real_{I,O,1}(\psi, \infty, 1)$. To formally prove this, we use Lemma 3 and show that $w\_real_{O,I,2}(\neg\psi, 1, \infty)$. Note that $\neg\psi = F((q \wedge X\neg p) \vee (\neg q \wedge Xp))$. Now, for every I/O-transducer $\mathcal{T}$, if $\mathcal{T}$ outputs q in its initial state, the O/I-1-transducer $\mathcal{T}'$ that always outputs $\neg p$ is such that the unique 2-computation of $\mathcal{T}'\|\mathcal{T}$ satisfies $\neg\psi$. Similarly, if $\mathcal{T}$ outputs $\neg q$ in its initial state, the corresponding O/I-1-transducer $\mathcal{T}'$ always outputs p.

On the other hand, $w\_real_{I,O,1}(\psi, \infty, 1)$. Indeed, there are two possible O/I-1-transducers: $\mathcal{T}_1'$ that always outputs p, and $\mathcal{T}_2'$ that always outputs $\neg p$. For $\mathcal{T}_1'$, the I/O-transducer $\mathcal{T}$ that always outputs q is such that the unique 1-computation of $\mathcal{T}\|\mathcal{T}_2'$ is $\{p, q\}^\omega$. For $\mathcal{T}_2'$, the I/O-transducer $\mathcal{T}$ that always outputs $\neg q$ is such that the unique 1-computation of $\mathcal{T}\|\mathcal{T}_2'$ is $\emptyset^\omega$. Since both satisfy ψ, we are done.

Example 5 shows that strong and weak realizability do not coincide in the bounded setting. We now generalize the example to all bounds, and show that it implies that the games that correspond to bounded synthesis are not determined. We first need some notations.

For two I/O-transducers $\mathcal{T}_1$ and $\mathcal{T}_2$, we say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are *equivalent*, denoted $\mathcal{T}_1 \equiv \mathcal{T}_2$, if $\mathcal{T}_1(w) = \mathcal{T}_2(w)$ for every word $w \in (2^I)^*$; otherwise, $\mathcal{T}_1$ and $\mathcal{T}_2$ are inequivalent. If $\mathcal{T}$ is a transducer and s an state of $\mathcal{T}$, then denote by $\mathcal{T}/s$ the transducer that is the same as $\mathcal{T}$ except that it has s as the start state. We say that $\mathcal{T}$ is *minimized* if there is no transducer $\mathcal{T}'$ such that $\mathcal{T} \equiv \mathcal{T}'$ and $\mathcal{T}'$ has strictly fewer states than $\mathcal{T}$. We say that two transducers $\mathcal{T}_1, \mathcal{T}_2$ are *structurally equivalent*, denoted $\mathcal{T}_1 \sim \mathcal{T}_2$, if (i) for every state s_1 of $\mathcal{T}_1$ there is a state s_2 of $\mathcal{T}_2$ such that $\mathcal{T}_1/s_1 \equiv \mathcal{T}_2/s_2$, and (ii) conversely, for every state s_2 of $\mathcal{T}_2$ there is a state s_1 of $\mathcal{T}_1$ such that $\mathcal{T}_1/s_1 \equiv \mathcal{T}_2/s_2$. Note that two minimized transducers are structurally equivalent iff they are isomorphic (but their start states do not need to map to each other in the isomorphism) [16].

Consider an I/O-transducer $\mathcal{T}$ with k states. A *checking sequence* for $\mathcal{T}$ is a word $w \in (2^I)^*$ such that for all transducers $\mathcal{T}'$ with at most k states, if $\mathcal{T}'(w) = \mathcal{T}(w)$ then $\mathcal{T}' \sim \mathcal{T}$. A transducer is *strongly connected* if every state can reach every other state. It is known that every strongly connected k-state transducer $\mathcal{T}$ has a checking sequence (of length at most exponential in k), and furthermore, if $\mathcal{T}$ is minimized, then it has one of length polynomial in k; see [16] for background and an overview on checking and other test sequences of transducers.

For a word $w \in (2^I)^*$, let $\theta(w)$ be an LTL formula over I that holds in exactly all computations in $(2^{I\cup O})^\omega$ whose prefix agrees with w on the signals in I. Thus, if $w = i_0, i_1, \ldots, i_l$, then $\theta(w) = \bigwedge_{0\leq j\leq l} X^j((\wedge_{x\in i_j} x) \wedge (\wedge_{x\notin i_j} \neg x))$. We define $\theta(y)$ similarly, for $y \in (2^O)^*$.

We now use checking sequences in order to show that bounding the sizes of the components, strong and weak synthesis no longer coincide, and determinacy fails.

▶ **Theorem 6.** *For every $k \geq 1$ there is an LTL formula ψ_k such that $w\_real_{I,O,1}(\psi_k, \infty, k)$ but not $s\_real_{I,O,1}(\psi_k, \infty, k)$; equivalently, neither $s\_real_{I,O,1}(\psi_k, \infty, k)$ nor $s\_real_{I,O,2}(\neg\psi_k, k, \infty)$.*

Proof. We give first a high level idea of the approach. We construct an LTL formula ψ_k that is weakly realizable, and is "almost strongly realizable", in the sense that the system can succeed in ensuring the formula by using one of two strategies (transducers): the first strategy works for all environment k-transducers except for those that are isomorphic to a specific one $\mathcal{T}_1'$, and the second strategy works for all environment k-transducers except for those isomorphic to a second one $\mathcal{T}_2'$. However, no system strategy works for both $\mathcal{T}_1', \mathcal{T}_2'$, and thus if the system does not know the environment transducer, then it cannot ensure ψ_k, i.e. ψ_k is not strongly realizable. In our construction, we pick suitable transducers $\mathcal{T}_1'$ and $\mathcal{T}_2'$ and construct from them a formula ψ_k that has the desired properties. Note that the presence of a bound on the size of the transducers is critical for this to be possible (and is obviously essential for the construction), i.e. there is no such formula ψ_∞, since weak and strong realizability coincide in the unbounded case.

We proceed now with the details of the construction. For an O/I transducer $\mathcal{T}_i'$ with k states that is minimized and strongly connected, let $y_i \in (2^O)^*$ be a checking sequence for $\mathcal{T}_i'$, let y_i/w_i be the 2-computation of $\mathcal{T}_i'$ on y_i, and let $\varphi_i = \theta(y_i) \wedge \neg\theta(w_i)$. Note that $s\_real_{I,O,1}(\varphi_i, \infty, k-1)$. To see why, consider an I/O-transducer $\mathcal{T}$ that ignores the input and outputs y_i. Since y_i is a checking sequence for $\mathcal{T}_i'$, and $\mathcal{T}_i'$ has k states and is minimized, the sequence y_i/w_i cannot be generated in an interaction with an O/I-transducer with at most k states that is not isomorphic with $\mathcal{T}_i'$. Hence, for every transducer $\mathcal{T}'$ with $k-1$ states, the 1-computation of $\mathcal{T}\|\mathcal{T}'$ satisfies both $\theta(y_i)$ and $\neg\theta(w_i)$. On the other hand, note also that $\neg s\_real_{I,O,1}(\varphi_i, \infty, k)$ and even $\neg w\_real_{I,O,1}(\varphi_i, \infty, k)$. Indeed, the O/I-transducer $\mathcal{T}_i'$ is such that for all I/O-transducers $\mathcal{T}$, if the 1-computation of $\mathcal{T}\|\mathcal{T}_i'$ satisfies $\theta(y_i)$, then it also satisfies $\theta(w_i)$.

We would like to use φ_i in order to construct a specification ψ_k as required. Let $\mathcal{T}_1'$ and $\mathcal{T}_2'$ be two nonisomorphic minimized strongly connected O/I transducers with k states; thus $\mathcal{T}_1'$, $\mathcal{T}_2'$ are not structurally equivalent (since they are minimized). Let o_1 and o_2 be two different letters in 2^O, let s_1 be the state that $\mathcal{T}_1'$ reaches when it reads o_1 from its initial state, and let s_2 be the state that $\mathcal{T}_2'$ reaches when it reads o_2 from its initial state. Finally, let y_1 be a checking sequence for the transducer $\mathcal{T}_1'/s_1$ (i.e., $\mathcal{T}_1'$ with initial state s_1) with corresponding output w_1, and let y_2 be a checking sequence for $\mathcal{T}_2'/s_2$ with corresponding output w_2. We define $\psi_k = (o_1 \wedge X(\theta(y_1) \wedge \neg\theta(w_1))) \vee (o_2 \wedge X(\theta(y_2) \wedge \neg\theta(w_2)))$. In the full version we prove that $w\_real_{I,O,1}(\psi_k, \infty, k)$ but $\neg s\_real_{I,O,1}(\psi_k, \infty, k)$. ◀

4 Bounded Systems

We start by studying the strong-realizability problem for bounded systems. This problem is first studied in [25]. The motivation there is to deal with the computational complexity of temporal synthesis: the complexity of deciding whether $real_{I,O,b}(\psi)$ holds is known to be 2EXPTIME-complete [22]. Furthermore, for each $n > 0$ one can construct an LTL formula φ_n of length $O(n)$ such that the smallest transducer $\mathcal{T}$ realizing φ_n has at least 2^{2^n} states [23]. In practice, therefore, we may want to bound the size of the systems under consideration, motivating us to ask whether $s\_real_{I,O,b}(\psi, k, \infty)$ holds, instead of asking whether $real_{I,O,b}(\psi)$ holds.

Note that the strong realizability problem for bounded systems has two parameters: a specification ψ and a bound k. Here we would like to understand the complexity with respect to k. Thus, we would like to "neutralize" here the effect of ψ on the complexity of checking whether $s\_real_{I,O,b}(\psi, k, \infty)$ holds. Fixing ψ would not help us, as, by the small model property, it induces a fixed bound on the size of a realizing transducer for ψ, if one exists. The complexity of the unbounded synthesis problem for LTL follows from the need to translate the LTL specification to a deterministic automaton on infinite words. Such a translation involves a doubly exponential blow-up [14], and is the source of the complexity of the synthesis problem. Indeed, for specifications given by means of nondeterministic or deterministic Büchi automata, the synthesis problem is complete in EXPTIME and PTIME, respectively [28].

Accordingly, we neutralize the dominance of the specification ψ by considering, instead of a specification given by an LTL formula, a specification given by a deterministic Büchi automaton $\mathcal{A}$ over the alphabet $2^{I \cup O}$. We denote classes of automata by acronyms in $\{D, N\} \times \{F, B\} \times \{W, T\}$. The first letter stands for the branching mode of the automaton (deterministic or nondeterministic); the second letter stands for the acceptance-condition type (finite words or Büchi); the third letter stands for the object over which the automaton runs (words or trees). For example, NFW stands for nondeterministic automata on finite words, and DBT stands for deterministic Büchi tree automata. We note that while DBWs are less expressive than NBWs, we still work with DBW rather than, say, deterministic parity word automata. The reason is that the nonemptiness problem for deterministic parity tree automata is not known to be polynomial, and we want to emphasize the fact that the hardness results we are going to prove are not due to the automaton and are due to k; the upper bounds we are going to present for DBWs are valid also for specifications given by a deterministic parity word automata.

Working with specifications that are automata, it is convenient to talk about alphabets Σ_I and Σ_O, where the transducers are Σ_I/Σ_O-transducers, in which the transitions are labeled by letters in Σ_I and the states are labeled by letters in Σ_O (or dually, are Σ_O/Σ_I-transducers). The alphabet of the specification DBW is then $\Sigma_O \times \Sigma_I$ when we specify 1-computations, and is $\Sigma_I \times \Sigma_O$ when we specify 2-computations.

▶ **Theorem 7.** *Deciding $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A}, k, \infty)$, for $b \in \{1, 2\}$, is NP-complete.*

The proof of Theorem 7 is given in the full version. The lower bound is by a simple reduction from the Vertex Cover problem: the states of the system correspond to the cover, and whenever the environment gives an edge of the graph to the system, the system should respond with the vertex that covers it.

By Lemma 3, we have that $w\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A},\infty,k)$ iff not $s\_real_{\Sigma_O,\Sigma_I,\tilde{b}}(\mathcal{A},k,\infty)$. Theorem 7 then immediately implies the following.

► **Corollary 8.** *Deciding $w\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A},\infty,k)$, for $b \in \{1,2\}$, is co-NP-complete.*

The implication of Theorem 7 and Corollary 8 is that deciding strong realizability with bounded systems (resp., weak realizability of bounded environments) amounts to search for a bounded system (resp., environment) that satisfies (resp., falsifies) the specification. Thus, what we have is essentially exhaustive search combined with model checking. The (co)-NP lower bounds tell us that there is no way of getting around the need to do an exhaustive search. Note that Corollary 8 already refers to the setting in which the environment is bounded, and it studies weak realizability there. In Section 5 below we study strong realizability in this setting.

5 Bounded Environments

We now turn to study the case of strong realizability in a setting of bounded environments (or, dually, weak realizability in a setting of bounded systems). In fact, bounding the environment is of interest already in the context of model-checking. For a system modeled by an I/O-transducer $\mathcal{T}$, a specification $\mathcal{A}$ given by a DBW, and $k \geq 1$, we say that $\mathcal{T}$ satisfies $\mathcal{A}$ with respect to k-environments if for all O/I-k-transducers $\mathcal{T}'$, the computation $\mathcal{T}\|\mathcal{T}'$ satisfies $\mathcal{A}$.

► **Theorem 9.** *Given an I/O-transducer $\mathcal{T}$, a DBW $\mathcal{A}$ over the alphabet $2^{I\cup O}$, and $k \geq 1$, the problem of deciding whether $\mathcal{T}$ satisfies $\mathcal{A}$ with respect to k-environments is co-NP-complete.*

The proof of Theorem 9 is given in the full version. The lower bound is by a reduction from the complement of the Hamiltonian Circle Problem [11].

We now turn to study the strong realizability problem when both the system and the environment are bounded.

► **Theorem 10.** *Deciding $s\_real_{\Sigma_I,\Sigma_O,1}(\mathcal{A},n,m)$ and $s\_real_{\Sigma_I,\Sigma_O,2}(\mathcal{A},n,m)$, for a given DBW $\mathcal{A}$ over alphabets Σ_I, Σ_O, and positive integers n, m in unary, is Σ_2^P-complete.*

Proof. We prove the claim for $b = 1$. The argument for $b = 2$ is analogous. The strong realizability property $s\_real_{\Sigma_I,\Sigma_O,1}(\mathcal{A},n,m)$ holds iff there exists a Σ_I/Σ_O transducer $\mathcal{T}$ with at most n states such that for every Σ_O/Σ_I transducer $\mathcal{T}'$ with at most m states, the 1-execution of $\mathcal{T}\|\mathcal{T}'$ satisfies the DBW specification $\mathcal{A}$. Membership in Σ_2^P follows from the fact that the sizes of the existentially and universally quantified transducers $\mathcal{T}$ and $\mathcal{T}'$ respectively are polynomially bounded in the size of the input, and the fact that we can check in polynomial time whether the 1-execution of $\mathcal{T}\|\mathcal{T}'$ satisfies $\mathcal{A}$ for given $\mathcal{T}, \mathcal{T}'$, and $\mathcal{A}$ [27].

To prove the hardness we reduce from the problem of deciding the truth of a formula of the form $\exists x \forall y \Phi(x,y)$, where x and y are vectors of Boolean variables and Φ is a formula in disjunctive normal form [11]. Let $x = (x_1,\ldots,x_k)$, $y = (y_1,\ldots y_k)$ (we assume without loss of generality that x, y have the same number k of variables), and let $\Phi = C_1 \vee C_2 \vee \ldots \vee C_p$, where each term C_i is a conjunction of literals (variables in x or y or their negations). We construct an instance $(\mathcal{A},n,m)$ of the strong realizability problem such that $s\_real_{\Sigma_I,\Sigma_O,1}(\mathcal{A},n,m)$ iff $\exists x \forall y \Phi(x,y)$.

We describe the reduction in detail in the full version. Here we describe the general idea. The integers n, m are both set to $2p+1+k$. The input and output alphabets have size $2p+1+2k$: The input alphabet is $\Sigma_I = \{d_0\} \cup \{d_i, d_i' | 1 \leq i \leq p\} \cup \{y_j, \bar{y}_j | 1 \leq j \leq k\}$, and the output alphabet is $\Sigma_O = \{c_0\} \cup \{c_i, c_i' | 1 \leq i \leq p\} \cup \{x_j, \bar{x}_j | 1 \leq j \leq k\}$.

Recall that the alphabet of the DBW $\mathcal{A}$ is $\Sigma_O \times \Sigma_I$ and it specifies the expected behavior of the system with respect to all input sequences. Thus, $\mathcal{A}$ should not limit the behavior of the environment. Since, however, the interaction between the system and the environment would be of interest only for some behaviors of the environment, we are going to describe how $\mathcal{A}$ prescribe an expected format of interaction for both the system and the environment, and we assume that if, during the interaction, the system deviates from this format then $\mathcal{A}$ moves to rejecting sink, and if the system follows the format but the environment deviates, then $\mathcal{A}$ accepts.

The DBW $\mathcal{A}$ prescribes the expected format in two phases. The goal of Phase 1 is to force the system and the environment to commit to a truth assignments for the variables in x and y, respectively, and to set aside states that output $c_1, c'_1, \ldots, c_p, c'_p$, and $d_1, d'_1, \ldots, d_p, d'_p$, respectively. Thus, if the system follows the prescribed format in Phase 1, then it outputs $2p + 1 + k$ different symbols during this phase: c_0, c_i, c'_i for $i = 1, \ldots, p$, and either x_j or $\bar{x}_j$ for $j = 1, \ldots, k$. Since the bound n on its number of states is $2p+1+k$, this implies that each state of the system has as output one of the letters above. In particular, this means that for each variable x_j, the system has exactly one corresponding state that outputs x_j or $\bar{x}_j$, which corresponds to assigning value true or false, respectively, to the variable x_j. Similarly, if the environment follows the prescribed format in Phase 1, and then it has exactly $2p + 1 + k$ states, corresponding to the n letters generated d_0, d_i, d'_i for $i = 1, \ldots, p$, and either y_j or $\bar{y}_j$ for $j = 1, \ldots, k$, and they induce a truth assignment for the y variables.

During Phase 2, the DBW $\mathcal{A}$ checks that the assignment to which the system and environment commit in Phase 1 satisfies Φ. Phase 2 consists of p stages, one for each term C_i of Φ. In the stage for C_i, the DBW $\mathcal{A}$ goes over the $2k$ variables and checks whether the assignment for them satisfies C_i. Recall that C_i is a conjunction. When $\mathcal{A}$ detects that the assignment contradicts a requirement imposed by C_i, it moves to the phase for C_{i+1}, or rejects, if $i = p$. When $\mathcal{A}$ concludes that the assignment satisfies C_i, it accepts. ◀

We can now turn to the problem of synthesis with bounded environments. We first need some definitions. For a word $w/y \in (\Sigma_I \times \Sigma_O)^*$ and $k \geq 1$, we say that w/y is *k-generable* if there is a Σ_I/Σ_O-transducer $\mathcal{T}$ with at most k states such that $\mathcal{T}(w) = w/y$. Let $L_k \subseteq (\Sigma_I \times \Sigma_O)^*$ be the set of k-generable words.

▶ **Lemma 11.** *There is a DFW with $2^{k^{O(k)}}$ states that recognizes L_k.*

Proof. We prove that there is an NFW with $k^{O(k)}$ states that recognizes L_k. The NFW guesses a Σ_I/Σ_O-transducer $\mathcal{T}$ with at most k states and then simulates it, checking that the y track is indeed the output of the w track. The number of Σ_I/Σ_O-transducers $\mathcal{T}$ with exactly k states is $k|\Sigma_O|^k k^{k|\Sigma_I|}$, and the number of transducers with at most k states, is no more than k times this expression. ◀

We can now solve the strong realizability problem for bounded environments.

▶ **Theorem 12.** *Deciding $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A}, \infty, k)$, for $b \in \{1, 2\}$ is in 2EXPTIME.*

Proof. We prove the claim for $b = 1$. The argument for $b = 2$ is analogous.

We recall first how ones decides $s\_real_{\Sigma_I,\Sigma_O,1}(\mathcal{A}, \infty, \infty)$. The key idea is to consider the following game between two players, called System and Environment. In each round, System first chooses a letter in Σ_I and Environment then chooses a letter in Σ_O. System wins if the play, which is the infinite word in $(\Sigma_I \times \Sigma_O)^\omega$ that results from the choices of System and Environment is accepted by $\mathcal{A}$. Checking if System wins the game is equivalent to testing nonemptiness of Büchi tree automata and can be done in quadratic time [29]. Furthermore, if System wins the game, then there is a strategy that depends only on the states of $\mathcal{A}$. Thus, if $\mathcal{A} = \langle \Sigma_I \times \Sigma_O, Q, q_0, \rho, F \rangle$, then the nonemptiness algorithm yields a function $L : Q \to \Sigma_O$, which means that $\mathcal{A}$ is strongly realized by the transducer $\mathcal{T} = \langle \Sigma_I, \Sigma_O, Q, q_0, \eta, L \rangle$, where $\eta(q, \sigma) = \rho(q, \langle \sigma, L(q) \rangle)$.

In deciding $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A},\infty,k)$, System only has to win against environments with at most k states. Thus, if a prefix of the play is not k-generable then System wins immediately, since no transducer with at most k states can generate such a play. Let $\mathcal{D}_k$ be a DFW that accepts exactly all words that are not k-generable. By Lemma 11, such a DFW with $2^{2^{O(k)}}$ states exists.

Then, $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A},\infty,k)$ holds iff $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A}\times\mathcal{D}_k,\infty,\infty)$, where $\mathcal{A}\times\mathcal{D}_k$ is the product of $\mathcal{A}$ and $\mathcal{D}_k$, which accepts a word $w \in (\Sigma_I \times \Sigma_O)^\omega$ if w is accepted by $\mathcal{A}$ or is not k-generable. It follows that if $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A},\infty,k)$, then $\mathcal{A}$ is realized by a transducer who states space is that of $\mathcal{A}\times\mathcal{D}_k$. ◀

Theorem 12 tells us that we can solve strong realizability against bounded transducers, but at a cost that is doubly exponential in k. Thus, on the one hand, we expect more specifications to be realizable when we bound the size of the adversaries, but, on the other hand, deciding such realizability comes at a considerable cost. The question is whether this cost is unavoidable. To prove this, we would have to show that deciding $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A},\infty,k)$ is 2EXPTIME-hard.
OPEN QUESTION: Is deciding $s\_real_{\Sigma_I,\Sigma_O,b}(\mathcal{A},\infty,k)$ 2EXPTIME-hard?

As in the case of bounded systems, we can show that the strong realizability problem for bounded environment is at least NP-hard. The problem, however, seems to be much harder. As supporting evidence for the hardness of the problem, we show that recognizing k-generable words by an automaton requires doubly-exponential size.

For a word $x = x_1, \dots x_n \in \Sigma_I^*$, a *combination lock* transducer for x is $\mathcal{T}_x = \langle \Sigma_I, \Sigma_O, \{0,\dots,n\}, 0, \tau\rangle$, where $\Sigma_O = \{0,1\}$, and for all $0 \le j < n$, we have $\rho(j, x_{j+1}) = j+1$ and $\rho(j,\sigma) = 0$, for all $\sigma \neq x_{j+1}$. Also, $\rho(n,\sigma) = n$, for all $\sigma \in \Sigma_I$. Finally, $\tau(j) = 0$ for all $0 \le j < n$ and $\tau(n) = 1$. Thus, $\mathcal{T}_x$ outputs 0 in all states but n. It can read x from its initial state, in which case it reaches the state n, where it outputs 1. When a violation of x is detected, $\mathcal{T}_x$ goes back to its initial state.

▶ **Theorem 13.** *A DFW that recognizes L_k has at least 2^{2^k} states.*

Proof. Recall that two words $x_1, x_2 \in (\Sigma_I \times \Sigma_O)^*$ are L_k-equivalent, in the Myhill-Nerode sense, iff for all $z \in (\Sigma_I \times \Sigma_O)^*$, we have that $x_1 \cdot z \in L_k$ iff $x_2 \cdot z \in L_k$. The number of states in a minimal DFW for L_k is the number of equivalence classes of the L_k-equivalence relation.

Let $\Sigma_I = \{a, b, \#\}$ and $\Sigma_O = \{0,1\}$. For a word $x \in (a+b)^k$, let $\mathcal{T}_x$ be the transducer obtained from the combination lock $\{a,b\}/\{0,1\}$-transducer for x by adding $\#$-transitions from all states to the initial state. Thus, $\#$ is an input reset symbol that resets to the initial state from all states. For every subset P of $(a+b)^k$, let $w(P)$ be the Σ_I/Σ_O word in which the input part consists of the words in P in some order, say lexicographic, with each word preceded by a reset, and the output part is all 0. We claim that all the words $w(P)$, for different subsets P, are not L_k-equivalent. Let P and Q be two different subsets. Assume, without loss of generality, that $x \in Q \setminus P$.

Note that the transducer T_x is strongly connected, and it is easy to see also that it is minimized. Let w be a checking sequence for $\mathcal{T}_x$, and let $\mathcal{T}_x(w) = w/y$. That is, the only k-state transducer that can generate w/y (including k-transducers that are not combination locks) is $\mathcal{T}_x$ starting from some state. If we append to $w(P)$ the word $\#/0 \cdot w/y$, then the resulting word is in L_k, as $\mathcal{T}_x$ generates it. On the other hand, if we append the word $\#/0 \cdot w/y$ to $w(Q)$, then the resulting word is not in L_k. Indeed, because of the w/y portion, the only k-transducer that could possibly generate the resulting word is $\mathcal{T}_x$, but since $x \in Q$, the transducer $\mathcal{T}_x$ cannot generate $w(Q)$, as it would output 1 after reading $\#x$. [4] ◀

[4] A small variant of the argument holds also for binary input alphabet, i.e., without the additional reset symbol. Restrict to combination locks where the first and the last symbol of the combination word is a, and replace the reset symbol by b^k. Then, b^k acts like a reset for these machines.

5.1 Absolute liveness properties

A specification ψ is of an *absolute liveness* property if for every computation π, we have that $\pi \models \psi$ iff $\pi \models F\psi$ [26]. In this section we show that for absolute liveness properties, weak and strong realizability against bounded environments coincide, and the complexity of deciding realizability is only exponential in the bound. Intuitively, it follows from the fact that the system can take its time to learn the environment with which it interacts, and then follow a strategy against that environment.

We formalize this intuition by means of the so-called *machine identification problem*: Given $k \geq 1$, we say that a word w is a *k-identifier* for every two O/I-k-transducers $\mathcal{T}_1', \mathcal{T}_2'$, if the two transducers produce the same output sequence y in response to w, and if s_1, s_2 are their states respectively after processing w, then $\mathcal{T}_1'/s_1 \equiv \mathcal{T}_2'/s_2$. In other words, observing the response of the environment to w, identifies uniquely up to equivalence the part of the environment transducer that is reachable from the final state *after* w. The machine identification problem was formulated and solved by Moore in his classical paper [19]. It is shown there that for every $k \geq 1$, there is a k-identifier of length exponential in k, and it can be constructed in exponential time. (The word w is essentially a homing sequence of the disjoint union of all k-transducers.)

▶ **Theorem 14.** *Let ψ be an absolute liveness specification. Then*

1. *$w\_real_{I,O,1}(\psi, \infty, k)$ iff $s\_real_{I,O,1}(\psi, \infty, k)$.*
2. *If ψ is given as a DBW, we can decide $s\_real_{I,O,1}(\psi, \infty, k)$ in time polynomial in ψ and exponential in k.*

Proof. Clearly $s\_real_{I,O,1}(\psi, \infty, k)$ implies $w\_real_{I,O,1}(\psi, \infty, k)$. For the other direction, assume that $w\_real_{I,O,1}(\psi, \infty, k)$ holds. Then, for each O/I-k-transducer $\mathcal{T}'$ there is an I/O-transducer $\mathcal{T}$ that guarantees that ψ holds. We need, however, one I/O-transducer $\mathcal{T}$ that can guarantee ψ against all O/I-k-transducers. What $\mathcal{T}$ can do is first output a k-identifier sequence w. After observing the response y of the k-transducer $\mathcal{T}'$ of the environment, we can construct a transducer $\mathcal{T}''$ with at most k states that is equivalent to $\mathcal{T}'/s$ where s is the current state of $\mathcal{T}'$ after w. Then, using weak realizability, $\mathcal{T}$ can simulate the I/O-transducer that wins against $\mathcal{T}''$. This proves the first claim.

The second claim follows from the fact that, for every given k-transducer $\mathcal{T}'$ for the environment, we can determine in polynomial time whether there is a system that satisfies a DBW specification ψ with environment $\mathcal{T}'$, hence $w\_real_{I,O,1}(\psi, \infty, k)$ (and $s\_real_{I,O,1}(\psi, \infty, k)$) can be decided in time polynomial in ψ and exponential in k. If $s\_real_{I,O,1}(\psi, \infty, k)$ holds, then a transducer $\mathcal{T}$ for the system that realizes ψ can be constructed also in time polynomial in ψ and exponential in k, as explained above. ◀

References

1 A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.

2 J.R. Büchi and L.H. Landweber. Solving sequential conditions by finite-state strategies. *Trans. AMS*, 138:295–311, 1969.

3 K. Chatterjee, T. Henzinger, and B. Jobstmann. Environment assumptions for synthesis. In *Proc. 19th CONCUR*, LNCS 5201, pages 147–161, 2008.

4 A. Church. Logic, arithmetics, and automata. In *Proc. Int. Congress of Mathematicians, 1962*, pages 23–35. Institut Mittag-Leffler, 1963.

5 E.M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *Proc. 32st DAC*, pages 427–432, 1995.

6 R. Ehlers. Symbolic bounded synthesis. In *Proc. 22nd CAV*, LNCS 6174, pages 365–379, 2010.

7 E.A. Emerson. Automata, tableaux, and temporal logics. In *Proc. Workshop on Logic of Programs*, LNCS 193, pages 79–87. Springer, 1985.

8 E. Filiot, N. Jin, and J.-F. Raskin. An antichain algorithm for LTL realizability. In *Proc. 21st CAV*, LNCS 5643, pages 263–277, 2009.

9 D. Fisman, O. Kupferman, and Y. Lustig. Rational synthesis. In *Proc. 16th TACAS*, LNCS 6015, pages 190–204. Springer, 2010.

10 M. Faella and M. Napoli and M. Parente. Graded Alternating-Time Temporal Logic In *Proc. 10th LPAR*, LNCS 6355, pages 192–211. Springer 2010.

11 M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. Freeman and Co., 1979.

12 A. Harding, M. Ryan, and P. Schobbens. A new algorithm for strategy synthesis in LTL games. In *Proc. 11th TACAS*, LNCS 3440, pages 477–492. Springer, 2005.

13 O. Kupferman and S. Sheinvald-Faragy. Finding shortest witnesses to the nonemptiness of automata on infinite words. In *Proc. 17th CONCUR*, LNCS 4137, pages 492–508. Springer, 2006.

14 O. Kupferman and M.Y. Vardi. From linear time to branching time. *ACM Transactions on Computational Logic*, 6(2):273–294, 2005.

15 O. Kupferman and M.Y. Vardi. Synthesis of trigger properties. In *Proc. 16th LPAR*,LNCS 6355, pages 312–331. Springer, 2010.

16 D. Lee and M. Yannakakis. Principles and methods of testing finite state machines - a survey. *Proc. IEEE*, 84:1089–1123, 1996.

17 D.A. Martin. Borel determinacy. *Annals of Mathematics*, 65:363–371, 1975.

18 A. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1996.

19 E.F. Moore. Gedanken-experiments on sequential machines. In *Automata studies*, volume 34 of *Ann. Math. Studies*, pages 129–153. Princeton Univ. Press, 1956.

20 C.H. Papadimitriou and M. Yannakakis. On complexity as bounded rationality (extended abstract). In *Proc. 26th STOC*, pages 726–733, 1994.

21 N. Piterman, A. Pnueli, and Y. Saar. Synthesis of reactive(1) designs. In *Proc. 7th VMCAI*, LNCS 3855, pages 364–380. Springer, 2006.

22 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. 16th POPL*, pages 179–190, 1989.

23 R. Rosner. *Modular Synthesis of Reactive Systems*. PhD thesis, Weizmann Inst. of Science, 1992.

24 A. Rubinstein. Finite automata play the repeated prisoner's dilemma. *J. Economic Theory*, 39(1):83 – 96, 1986.

25 S. Schewe and B. Finkbeiner. Bounded synthesis. In *Proc. 5th ATVA*, LNCS 4762, pages 474–488. Springer, 2007.

26 A.P. Sistla. Satefy, liveness and fairness in temporal logic. *Formal Aspects of Computing*, 6:495–511, 1994.

27 M.Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Logics for Concurrency: Structure versus Automata*, LNCS 1043, pages 238–266. Springer, 1996.

28 M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. 1st LICS*, pages 332–344, 1986.

29 M.Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *Journal of Computer and Systems Science*, 32(2):182–221, 1986.

30 G. Wang, A. Mishchenko, R. Brayton, and A. Sangiovanni-Vincentelli. Synthesizing FSMs according to co-Büchi properties. Technical report, UC Berkeley, 2005.

Linear temporal logic for regular cost functions

Denis Kuperberg

LIAFA/**CNRS/Université Paris 7, Denis Diderot, France**

Abstract

Regular cost functions have been introduced recently as an extension to the notion of regular languages with counting capabilities, which retains strong closure, equivalence, and decidability properties. The specificity of cost functions is that exact values are not considered, but only estimated.

In this paper, we define an extension of Linear Temporal Logic (LTL) over finite words to describe cost functions. We give an explicit translation from this new logic to automata. We then algebraically characterize the expressive power of this logic, using a new syntactic congruence for cost functions introduced in this paper.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.627

1 Introduction

Since the seminal works of Kleene and Rabin and Scott, the theory of regular languages is one of the cornerstones in computer science. Regular languages have many good properties, of closure, of equivalent characterizations, and of decidability, which makes them central in many situations.

Recently, the notion of regular cost function for words has been presented as a candidate for being a quantitative extension to the notion of regular languages, while retaining most of the fundamental properties of the original theory such as the closure properties, the various equivalent characterizations, and the decidability [2]. A cost function is an equivalence class of the functions from the domain (words in our case) to $\mathbb{N} \cup \{\infty\}$, modulo an equivalence relation $\approx$ which allows some distortion, but preserves the boundedness property over each subset of the domain. The model is an extension to the notion of languages in the following sense: one can identify a language with the function mapping each word inside the language to 0, and each word outside the language to ∞. It is a strict extension since regular cost functions have counting capabilities, e.g., counting the number of occurrences of letters, measuring the length of intervals, etc...

Linear Temporal Logic (LTL), which is a natural way to describe logical constraints over a linear structure, have also been a fertile subject of study, particularly in the context of regular languages and automata [10]. Moreover quantitative extensions of LTL have recently been successfully introduced. For instance the model Prompt-LTL introduced in [8] is interested in bounding the waiting time of all requests of a formula, and in this sense is quite close to the aim of cost functions.

In this paper, we extend LTL (over finite words) into a new logic with quantitative features (LTL$^{\leq}$), in order to describe cost functions over finite words with logical formulae. We do this by adding a new operator $U^{\leq N}$: a formula $\phi U^{\leq N} \psi$ means that ψ holds somewhere in the future, and ϕ has to hold until that point, except at most N times (we allow at most N "mistakes" of the until formula).

Related works and motivating examples

Regular cost functions are the continuation of a sequence of works that intend to solve difficult questions in language theory. Among several other decision problems, the most prominent example is the star-height problem: given a regular language L and an integer k, decide whether L can be expressed using a regular expression using at most k-nesting of Kleene stars. The problem was

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 627–636
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

resolved by Hashigushi [5] using a very intricate proof, and later by Kirsten [7] using an automaton that has counting features.

Finally, also using ideas inspired from [1], the theory of those automata over words has been unified in [2], in which cost functions are introduced, and suitable models of automata, algebra, and logic for defining them are presented and shown equivalent. Corresponding decidability results are provided. The resulting theory is a neat extension of the standard theory of regular languages to a quantitative setting.

On the logic side, Prompt-LTL, introduced in [8], is an interesting way to extend LTL in order to look at boundedness issues, and already gave interesting decidability and complexity results. Prompt-LTL would correspond in the framework of regular cost functions to a subclass of temporal cost functions introduced in [3]; in particular it is weaker than LTL$^{\leq}$ introduced here.

Contributions

It is known from [2] that regular cost functions are the ones recognizable by stabilization semigroups (or in an equivalent way, stabilization monoids), and from [3] than there is an effective quotient-wise minimal stabilization semigroup for each regular cost function. This model of semigroups extends the standard approach for languages.

We introduce a quantitative version of LTL in order to describe cost functions by means of logical formulas. The idea of this new logic is to bound the number of "mistakes" of Until operators, by adding a new operator $U^{\leq N}$. The first contribution of this paper is to give a direct translation from LTL$^{\leq}$-formulas to B-automata, which is an extension of the classic translation from LTL to Büchi automaton for languages. This translation preserves exact values (i.e. not only cost functions equivalence), which could be interesting in terms of future applications. We then show that regular cost functions described by LTL formulae are the same as the ones computed by aperiodic stabilization semigroups, and this characterization is effective. The proof uses a syntactic congruence for cost functions, introduced in this paper.

This work validates the algebraic approach for studying cost functions, since the analogy extends to syntactic congruence. It also allows a more user-friendly way to describe cost functions, since LTL can be more intuitive than automata or stabilization semigroups to describe a given cost function.

As it was done in [3] for temporal cost functions, the characterization result obtained here for LTL$^{\leq}$-definable cost functions follows the spirit of Schützenberger's theorem which links star-free languages with aperiodic monoids [9].

Organisation of the paper

After some notations, and reminder on cost functions, we introduce in Section 3 LTL$^{\leq}$ as a quantitative extension of LTL, and give an explicit translation from LTL$^{\leq}$-formulae to B-automata. We then present in Section 4 a syntactic congruence for cost functions, and show that it indeed computes the minimal stabilization semigroup of any regular cost function. We finally use this new tool to show that LTL$^{\leq}$ has the same expressive power as aperiodic stabilization semigroups.

Notations

We will note $\mathbb{N}$ the set of non-negative integers and $\mathbb{N}_\infty$ the set $\mathbb{N}\cup\{\infty\}$, ordered by $0 < 1 < \cdots < \infty$. If E is a set, $E^{\mathbb{N}}$ is the set of infinite sequences of elements of E (we will not use here the notion of infinite words). Such sequences will be denoted by bold letters ($\vec{a}$, $\vec{b}$,...). We will work with a fixed finite alphabet $\mathbb{A}$. The set of words over $\mathbb{A}$ is $\mathbb{A}^*$ and the empty word will be noted ε. The concatenation of words u and v is uv. The length of u is $|u|$. The number of occurrences of letter a

in u is $|u|_a$. Functions $\mathbb{N} \to \mathbb{N}$ will be denoted by letters $\alpha, \beta, \dots$, and will be extended to $\mathbb{N} \cup \{\infty\}$ by $\alpha(\infty) = \infty$.

2 Regular Cost functions

2.1 Cost functions and equivalence

If $L \subseteq \mathbb{A}^*$, we will note χ_L the function defined by $\chi_L(u) = 0$ if $u \in L$, ∞ if $u \notin L$. Let $\mathcal{F}$ be the set of functions : $\mathbb{A}^* \to \mathbb{N}_\infty$. For $f, g \in \mathcal{F}$ and α a function (see Notations), we say that $f \leq_\alpha g$ if $f \leq \alpha \circ g$, and $f \approx_\alpha g$ if $f \leq_\alpha g$ and $g \leq_\alpha f$. Finally $f \approx g$ if $f \approx_\alpha g$ for some α. This equivalence relation doesn't pay attention to exact values, but preserves the existence of bounds.

A *cost function* is an equivalence class of $\mathcal{F}/\approx$. Cost functions are noted $f, g, \dots$, and in practice they will be always be represented by one of their elements in $\mathcal{F}$.

2.2 B-automata

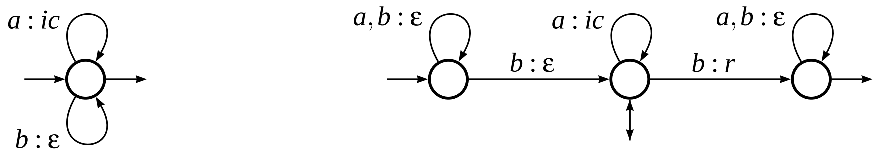
A B-automaton is a tuple $\langle Q, \mathbb{A}, In, Fin, \Gamma, \Delta\rangle$ where Q is the set of states, $\mathbb{A}$ the alphabet, In and Fin the sets of initial and final states, Γ the set of counters, and $\Delta \subseteq Q \times \mathbb{A} \times (\{i,r,c\}^*)^\Gamma \times Q$ is the set of transitions.

Counters have integers values starting at 0, and an action $\sigma \in (\{i,r,c\}^*)^\Gamma$ performs a sequence of atomic actions on each counter, where atomic actions are either i (increment by 1), r (reset to 0) or c (check the value). In particular we will note ε the action corresponding to the empty word : doing nothing on every counter. If e is a run, let $C(e)$ be the set of values checked during e on all counters of Γ.

A B-automaton $\mathcal{A}$ computes a regular cost function $[\![\mathcal{A}]\!]$ via the following semantic : $[\![\mathcal{A}]\!](u) = \inf\{\sup C(e), e \text{ run of } \mathcal{A} \text{ over } u\}$.

With the usual conventions that $\sup\emptyset = 0$ and $\inf\emptyset = \infty$. There exists also a dual model of B-automata, namely S-automata, that has the same expressive power, but we won't develop this further in this paper. See [2] for more details.

▶ Example 1. Let $\mathbb{A} = \{a,b\}$. The cost function $|\cdot|_a$ is the same as $2|\cdot|_a + 5$, it is computed by the following one-counter B-automaton on the left-hand side. The cost function $u \mapsto \min\{n \in \mathbb{N},\ a^n$ factor of $u\}$ is computed by the nondeterministic one-counter B-automaton on the right-hand side.



Moreover, as in the case of languages, cost functions can be recognized by an algebraic structure that extends the classic notion of semigroups, called stabilization semigroups. A stabilization semigroup $\mathbf{S} = \langle S, \cdot, \leq, \sharp\rangle$ is a partially ordered set S together with an internal binary operation $\cdot$ and an internal unary operation $a \mapsto a^\sharp$ defined only on idempotent elements (elements a such that $a \cdot a = a$). The formalism is quite heavy, see appendix for all details on axioms of stabilization semigroups and recognition of regular cost functions.

3 Quantitative LTL

We will now use an extension of LTL to describe some regular cost functions. This has been done successfully with regular languages, so we aim to obtain the same kind of results. Can we still go efficiently from an LTL-formula to an automaton?

3.1 Definition

The first thing to do is to extend LTL so that it can decribe cost functions instead of languages. We must add quantitative features, and this will be done by a new operator $U^{\leq N}$. Unlike in most uses of LTL, we work here over finite words.

Formulas of LTL$^{\leq}$ (on finite words on an alphabet $\mathbb{A}$) are defined by the following grammar :

$$\phi := a \mid \phi \wedge \phi \mid \phi \vee \phi \mid X\phi \mid \phi U \phi \mid \ \phi U^{\leq N} \phi \mid \Omega$$

Note the absence of negation in the definition of LTL$^{\leq}$. The negations have been pushed to the leaves.

- a means that the current letter is a, $\wedge$ and $\vee$ are the classic conjunction and disjunction;
- $X\phi$ means that ϕ is true at the next letter;
- $\phi U \psi$ means that ψ is true somewhere in the future, and ϕ holds until that point;
- $\phi U^{\leq N} \psi$ means that ψ is true somewhere in the future, and ϕ can be false at most N times before ψ. The variable N is unique, and is shared by all occurrences of $U^{\leq N}$ operator;
- Ω means that we are at the end of the word.

We can define $\top = (\bigvee_{a \in \mathbb{A}} a) \vee \Omega$ and $\bot = \neg\top$, meaning respectively true and false, and $\neg a = (\bigvee_{b \neq a} b) \vee \Omega$ to signify that the current letter is not a.

We also define connectors "eventually" : $F\varphi = \top U \varphi$ and "globally" : $G\varphi = \varphi U \Omega$.

3.2 Semantics

We want to associate a cost function $[\![\phi]\!]$ on words to any LTL$^{\leq}$-formula ϕ.

We will say that $u, n \models \phi$ (u, n is a model of ϕ) if ϕ is true on u with n as valuation for N, i.e. as number of errors for all the $U^{\leq N}$'s in the formula ϕ. We finally define

$$[\![\phi]\!](u) = \inf\{n \in \mathbb{N} / u, n \models \phi\}$$

We can remark that if $u, n \models \phi$, then for all $k \geq n, u, k \models \phi$, since the $U^{\leq N}$ operators appear always positively in the formula (that is why we don't allow the negation of an LTL$^{\leq}$-formula in general). In particular, $[\![\phi]\!](u) = 0$ means that $\forall n \in \mathbb{N}, u, n \models \phi$, and $[\![\phi]\!](u) = \infty$ means that $\forall n \in \mathbb{N}, u, n \not\models \phi$ (since $\inf \emptyset = \infty$).

▶ Proposition 2.

- $[\![a]\!](u) = 0$ if $u \in a\mathbb{A}^*$, and ∞ otherwise
- $[\![\Omega]\!](u) = 0$ if $u = \varepsilon$, and ∞ otherwise
- $[\![\phi \wedge \psi]\!] = \max([\![\phi]\!], [\![\psi]\!])$, and $[\![\phi \vee \psi]\!] = \min([\![\phi]\!], [\![\psi]\!])$
- $[\![X\phi]\!](au) = [\![\phi]\!](u)$, $[\![X\phi]\!](\varepsilon) = \infty$
- $[\![\top]\!] = 0$, and $[\![\bot]\!] = \infty$

▶ Example 3. Let $\phi = (\neg a) U^{\leq N} \Omega$, then $[\![\phi]\!] = |\cdot|_a$

We use LTL$^{\leq}$-formulae in order to describe cost functions, so we will always work modulo cost function equivalence $\approx$.

▶ Remark 4. If ϕ does not contain any operator $U^{\leq N}$, ϕ is a classic LTL-formula computing a language L, and $[\![\phi]\!] = \chi_L$.

3.3 From LTL$^{\leq}$ to B-Automata

We will now give a direct translation from LTL$^{\leq}$-formula to B-automata, i.e. given an LTL$^{\leq}$-formula ϕ on a finite alphabet $\mathbb{A}$, we want to build a B-automaton recognizing $[\![\phi]\!]$. This construction is adapted from the classic translation from LTL-formula to Büchi automata [4].

Let ϕ be an LTL$^{\leq}$-formula. We define $\mathrm{sub}(\phi)$ to be the set of subformulae of ϕ, and $Q = 2^{\mathrm{sub}(\phi)}$ to be the set of subsets of $\mathrm{sub}(\phi)$.

We want to define a B-automaton $\mathcal{A}_\phi = \langle Q, \mathbb{A}, In, Fin, \Gamma, \Delta\rangle$ such that $[\![\mathcal{A}]\!]_B \approx [\![\phi]\!]$.

We set the initial states to be $In = \{\{\phi\}\}$ and the final ones to be $Fin = \{\emptyset, \{\Omega\}\}$ We choose as set of counters $\Gamma = \{\gamma_1, \dots, \gamma_k\}$ where k is the number of occurences of the $U^{\leq N}$ operators in ϕ, labeled from $U_1^{\leq N}$ to $U_k^{\leq N}$.

A state is basically the set of constraints we have to verify before the end of the word, so the only two accepting states are the one with no constraint, or with only constraint to be at the end of the word.

The following definitions are the same as for the classical case (LTL to Büchi automata) :

▶ Definition 5.
- An atomic formula is either a letter $a \in \mathbb{A}$ or Ω
- A set Z of formulae is consistent if there is at most one atomic formula in it.
- A reduced formula is either an atomic formula or a Next formula (of the form $X\varphi$).
- A set Z is reduced if all its elements are reduced formulae.
- If Z is consistent and reduced, we define $\mathrm{next}(Z) = \{\varphi / X\varphi \in Z\}$.

▶ Lemma 6 (Next Step). If Z is consistent and reduced, for all $u \in \mathbb{A}^*, a \in \mathbb{A}$ and $n \in \mathbb{N}$,

$$au, n \models \bigwedge Z \text{ iff } u, n \models \bigwedge \mathrm{next}(Z) \text{ and } Z \cup \{a\} \text{ consistent}$$

We would like to define $\mathcal{A}_\phi$ with $Z \longrightarrow \mathrm{next}(Z)$ as transitions.

The problem is that $\mathrm{next}(Z)$ is not consistent and reduced in general. If $\mathrm{next}(Z)$ is inconsistent we remove it from the automaton. If it is consistent, we need to apply some reduction rules to get a reduced set of formulae. This consists in adding ε-transitions (but with possible actions on the counter) towards intermediate sets which are not actual states of the automaton (we will call them "pseudo-states"), until we reach a reduced set.

Let ψ be maximal (in size) not reduced in Y, we add the following transitions

- If $\psi = \varphi_1 \wedge \varphi_2$: $Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1, \varphi_2\}$
- If $\psi = \varphi_1 \vee \varphi_2$: $\begin{cases} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1\} \\ Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_2\} \end{cases}$
- If $\psi = \varphi_1 U \varphi_2$: $\begin{cases} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1, X\psi\} \\ Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_2\} \end{cases}$
- If $\psi = \varphi_1 U_j^{\leq N} \varphi_2$: $\begin{cases} Y \xrightarrow{\varepsilon:\varepsilon} Y \setminus \{\psi\} \cup \{\varphi_1, X\psi\} \\ Y \xrightarrow{\varepsilon:ic_j} Y \setminus \{\psi\} \cup \{X\psi\} \text{ (we count one mistake)} \\ Y \xrightarrow{\varepsilon:r_j} Y \setminus \{\psi\} \cup \{\varphi_2\} \end{cases}$

where action r_j (resp. ic_j) perform r (resp. ic) on counter γ_j and ε on the other counters.
The pseudo-states don't (a priori) belong to $Q = 2^{\mathrm{sub}(\phi)}$ because we add formulae $X\psi$ for $\psi \in \mathrm{sub}(\phi)$, so if Z is a reduced pseudo-state, $\mathrm{next}(Z)$ will be in Q again since we remove the new next operators.

The transitions of automaton $\mathcal{A}_\phi$ will be defined as follows:

$$\Delta = \left\{ Y \xrightarrow{a:\sigma} \mathrm{next}(Z) \mid Y \in Q, Z \cup \{a\} \text{ consistent and reduced}, Y \xrightarrow{\varepsilon:\sigma}_* Z \right\}$$

where $Y \xrightarrow{\varepsilon:\sigma}_{*} Z$ means that there is a sequence of ε-transitions from Y to Z with σ as combined action on counters.

▶ Definition 7. If σ is a sequence of actions on counters, we will call $\mathrm{val}(\sigma)$ the maximal value checked on a counter during σ with 0 as starting value of the counters, and $\mathrm{val}(\sigma) = 0$ if there is no check in σ. It corresponds to the value of a run of a B-automaton with σ as combined action of the counter.

▶ Lemma 8. Let $u = a_1 \dots a_m$ be a word on $\mathbb{A}$ and $Y_0 \overset{a_1:\sigma_1}{\to} Y_1 \overset{a_2:\sigma_2}{\to} \dots \overset{a_m:\sigma_m}{\to} Y_m$ an accepting run of $\mathcal{A}_\phi$.

Then for all $\psi \in \mathrm{sub}(\phi)$, for all $n \in \{0,\dots,m\}$, for all $Y_n \overset{\varepsilon:\sigma}{\to}_{*} Y \overset{\varepsilon:\sigma'}{\to}_{*} Z$ with $Z \cup \{a_{n+1}\}$ consistent and reduced, and $Y_{n+1} = \mathrm{next}(Z)$

$$\psi \in Y \implies a_{n+1}a_{n+2}\dots a_m, N \models \psi$$

where $N = \mathrm{val}(\sigma'\sigma_{n+1}\dots\sigma_m)$.

Lemma 8 implies the correctness of the automaton $\mathcal{A}_\phi$:
Let $Y_0 \overset{a_1:\sigma_1}{\to} Y_1 \overset{a_2:\sigma_2}{\to} \dots \overset{a_m:\sigma_m}{\to} Y_m$ be a valid run of $\mathcal{A}_\phi$ on u of value $N = [\![\mathcal{A}_\phi]\!]_B$, applying Lemma 8 with $n = 0$ and $Y = Y_0 = \{\phi\}$ gives us $u, N \models \phi$. Hence $[\![\phi]\!] \leq [\![\mathcal{A}_\phi]\!]_B$.

Conversely, let $N = [\![\phi]\!](u)$, then $u, N \models \phi$ so by definition of $\mathcal{A}_\phi$, it is straightforward to verify that there exists an accepting run of $\mathcal{A}_\phi$ over u of value $\leq N$ (each counter γ_i doing at most N mistakes relative to operator $U_i^{\leq N}$). Hence $[\![\mathcal{A}_\phi]\!]_B \leq [\![\phi]\!]$.

We finally get $[\![\mathcal{A}_\phi]\!]_B = [\![\phi]\!]$, the automaton $\mathcal{A}_\phi$ computes indeed the exact value of function $[\![\phi]\!]$ (and so we have obviously $[\![\mathcal{A}_\phi]\!]_B \approx [\![\phi]\!]$).

4 Algebraic characterization

We remind that as in the case of languages, stabilization semigroups recognize exactly regular cost functions, and there exists a quotient-wise minimal stabilization semigroup for each regular cost function [3].

In standard theory, it is equivalent for a regular language to be described by an LTL-formula, or to be recognized by an aperiodic semigroup. Is it still the case in the framework of regular cost functions? To answer this question we first need to develop a little further the algebraic theory of regular cost functions.

4.1 Syntactic congruence

In standard theory of languages, we can go from a description of a regular language L to a description of its syntactic monoid via the syntactic congruence. Moreover, when the language is not regular, we get an infinite monoid, so this equivalence can be used to “test” regularity of a language.

The main idea behind this equivalence is to identify words u and v if they “behave the same” relatively to the language L, i.e. L cannot separate u from v in any context : $\forall (x,y), xuy \in L \Leftrightarrow xvy \in L$.

The aim here is to define an analog to the syntactic congruence, but for regular cost functions instead of regular languages. Since cost functions look at quantitative aspects of words, the notions of "element" and "context" have to contain quantitative information : we want to be able to say things like “words with a lot of a’s behave the same as words with a few a’s”.

That is why we won’t define our equivalence over words, but over ♯-expressions, which are a way to describe words with quantitative information.

4.2 $\sharp$-expressions

We first define general $\sharp$-expressions as in [6] and [3] by just adding an operator $\sharp$ to words in order to repeat a subexpression "a lot of times". This differs from the stabilization monoid definition, in which the $\sharp$-operator can only be applied to specific elements (idempotents).

The set Expr of $\sharp$-expressions on an alphabet $\mathbb{A}$ is defined as follows:

$$e := a \in \mathbb{A} \mid ee \mid e^\sharp$$

If we choose a stabilization semigroup $\mathbf{S} = \langle S, \cdot, \leq, \sharp \rangle$ together with a function $h : \mathbb{A} \to S$, the eval function (from Expr to $\mathbf{S}$) is defined inductively by $\mathrm{eval}(a) = h(a), \mathrm{eval}(ee') = \mathrm{eval}(e) \cdot \mathrm{eval}(e')$, and $\mathrm{eval}(e^\sharp) = \mathrm{eval}(e)^\sharp$ ($\mathrm{eval}(e)$ has to be idempotent). We say that e is *well-formed for* $\mathbf{S}$ if $\mathrm{eval}(e)$ exists. Intuitively, it means that $\sharp$ was applied to subexpressions that corresponds to idempotent elements in $\mathbf{S}$.

If f is a regular cost function, e is *well-formed for* f iff e is well-formed for the minimal stabilization semigroup of f.

▶ Example 9. Let f be the cost function defined over $\{a\}^*$ by

$$f(a^n) = \begin{cases} n & \text{if } n \text{ even} \\ \infty & \text{otherwise} \end{cases}$$

The minimal stabilization semigroup of f is : $\{a, aa, (aa)^\sharp, (aa)^\sharp a\}$, with $aa \cdot a = a$ and $(aa)^\sharp a \cdot a = (aa)^\sharp$. Hence the $\sharp$-expression $aaa(aa)^\sharp$ is well-formed for f but the $\sharp$-expression $a^\sharp$ is not.

The $\sharp$-expressions that are not well-formed have to be removed from the set we want to quotient, in order to get only real elements of the syntactic semigroup.

4.3 $\omega\sharp$-expressions

We have defined the set of $\sharp$-expressions that we want to quotient to get the syntactic equivalence of cost functions. However, we saw that some of these $\sharp$-expressions may not be well-typed for the cost function f we want to study, and therefore does not correspond to an element in the syntactic stabilization semigroup of f.

Thus we need to be careful about the stabilization operator, and apply it only to "idempotent $\sharp$-expressions". To reach this goal, we will add an "idempotent operator" ω on $\sharp$-expressions, which will always associate an idempotent element (relative to f) to a $\sharp$-expression, so that we can later apply $\sharp$ and be sure of creating well-formed expressions for f.

We define the set Oexpr of $\omega\sharp$-expressions on an alphabet $\mathbb{A}$:

$$E := a \in \mathbb{A} \mid EE \mid E^\omega \mid E^{\omega\sharp}$$

The intuition behind operator ω is that x^ω is the idempotent obtained by iterating x (which always exists in finite semigroups).

A *context* $C[x]$ is a $\omega\sharp$-expression with possible occurrences of a free variable x. Let E be a $\omega\sharp$-expression, $C[E]$ is the $\omega\sharp$-expression obtained by replacing all occurrences of x by E in $C[x]$, i.e. $C[E] = C[x][x \leftarrow E]$. Let C_{OE} be the set of contexts on $\omega\sharp$-expressions.

We will now formally define the semantic of operator ω, and use $\omega\sharp$-expressions to get a syntactic equivalence on cost functions, without mistyped $\sharp$-expressions.

▶ Definition 10. If $E \in$ Oexpr and $k, n \in \mathbb{N}$, we define $E(k,n)$ to be the word $E[\omega \leftarrow k, \sharp \leftarrow n]$, where the exponential is relative to concatenation of words.

▶ Lemma 11. Let f be a regular cost function, there exists $K_f \in \mathbb{N}$ such that for any $E \in$ Oexpr, the $\sharp$-expression $E[\omega \leftarrow K_f!]$ is well-formed for f, and we are in one of these two cases

1. $\forall k \geq K_f, \{f(E(k!,n)), n \in \mathbb{N}\}$ is bounded : we say that $E \in f^B$.
2. $\forall k \geq K_f, \lim_{n\to\infty} f(E(k!,n)) = \infty$: we say that $E \in f^\infty$.

Proof. The proof is a little technical, since we have to reuse the definition of recognization by stabilization semigroup. K_f can simply be taken to be the size of the minimal stabilization semigroup of f. ◀

Here, f^B and f^∞ are the analogs for regular cost functions of "being in L" and "not being in L" in language theory. But this notion is now asymptotic, since we look at boundedness properties of quantitative information on words. Moreover, f^∞ and f^B are only defined here for regular cost functions, since K_f might not exist if f is not regular.

▶ Definition 12. Let f be a regular cost function, we write $E \rightleftharpoons_f E'$ if $(E \in f^B \Leftrightarrow E' \in f^B)$. Finally we define

$$E \equiv_f E' \text{ iff } \forall C[x] \in \mathrm{C_{OE}}, C[E] \rightleftharpoons_f C[E']$$

▶ Remark 13. If $u,v \in \mathbb{A}^*$, and L is a regular language, then $u \sim_L v$ iff $u \equiv_{\chi_L} v$ ($\sim_L$ being the syntactic congruence of L). In this sense, $\equiv$ is an extension of the classic syntactic congruence on languages.

Now that we have properly defined the equivalence $\equiv_f$ over Oexpr, it remains to verify that it is indeed a good syntactic congruence, i.e. Oexpr$/\equiv_f$ is the syntactic stabilization semigroup of f.

Indeed if f is a regular cost function, let $\mathbf{S}_f =$ Oexpr$/\equiv_f$. We can provide $\mathbf{S}_f$ with a structure of stabilization semigroup $\langle \mathbf{S}_f, \cdot, \leq, \sharp \rangle$.

▶ **Theorem 14.** *$\mathbf{S}_f$ is the minimal stabilization semigroup recognizing f.*

The proof consists basically in a bijection between classes of Oexpr for $\equiv_f$, and elements of the minimal stabilization semigroup as defined in appendix A.7 of [3].

4.4 Expressive power of LTL$^{\leq}$

If f is a regular cost function, we will call $\mathbf{S}_f$ the syntactic stabilization semigroup of f.

A finite semigroup $\mathbf{S} = \langle S, \cdot \rangle$ is called *aperiodic* if $\exists k \in \mathbb{N}, \forall s \in \mathbf{S}, s^{k+1} = s^k$. The definition is the same if $\mathbf{S}$ is a finite stabilization semigroup.

▶ Remark 15. For a regular cost function f, the statements "f is recognized by an aperiodic stabilization semigroup" and "$\mathbf{S}_f$ is aperiodic" are equivalent, since $\mathbf{S}_f$ is a quotient of all stabilization semigroups recognizing f.

▶ **Theorem 16.** *Let f be a cost function described by a LTL$^{\leq}$-formula, then f is regular and the syntactic stabilization semigroup of f is aperiodic.*

The proof of this theorem will be the first framework to use the syntactic congruence on cost functions.

If ϕ is a LTL$^{\leq}$-formula, we will say that ϕ verifies property AP if there exists $k \in \mathbb{N}$ such that for any $\omega\sharp$-expression E, $E^k \equiv_{[\![\phi]\!]} E^{k+1}$, which is equivalent to "$[\![\phi]\!]$ has an aperiodic syntactic stabilization semigroup".

With this in mind, we can do an induction on LTL$^{\leq}$-formulaes : we first show that $\mathbf{S}_\Omega$ and all $\mathbf{S}_a$ for $a \in \mathbb{A}$ are aperiodic.

We then proceed to the induction on ϕ : assuming that φ and ψ verify property AP, we show that $X\psi$, $\varphi \vee \psi$, $\varphi \wedge \psi$, $\varphi U \psi$ and $\varphi U^{\leq N}\psi$ verify property AP.

▶ **Theorem 17.** *Let f be a cost function recognized by an aperiodic stabilization semigroup, then f can be described by an LTL$^{\leq}$-formula.*

The proof of this theorem is a generalization of the proof of Wilke for aperiodic languages in [11]. However difficulties inherent to quantitative notions appear here.

The main issue comes from the fact that in the classical setting, computing the value of a word in a monoid returns a single element. This fact is used to do an induction on the size of the monoid, by considering the set of possible results as a smaller monoid. The problem is that with cost functions, there is some additional quantitative information, and we need to associate a sequence of elements of a stabilization monoid to a single word. Therefore, it requires some technical work to come back to a smaller stabilization monoid from these sequences.

▶ Corollary 18. The class of LTL$^{\leq}$-definable cost functions is decidable.

Proof. Theorems 16 and 17 imply that it is equivalent for a regular cost function to be LTL$^{\leq}$-definable or to have an aperiodic syntactic stabilization semigroup. If f is given by an automaton or a stabilization semigroup, we can compute its syntactic stabilization semigroup $\mathbf{S}_f$ (see [3]) and decide if f is LTL$^{\leq}$-definable by testing aperiodicity of $\mathbf{S}_f$. This can be done simply by iterating at most $|\mathbf{S}_f|$ times all elements of $\mathbf{S}_f$ and see if each element a reaches an element a^k such that $a^{k+1} = a^k$. ◀

5 Conclusion

We first defined LTL$^{\leq}$ as a quantitative extension of LTL. We started the study of LTL$^{\leq}$ by giving an explicit translation from LTL$^{\leq}$-formulae to B-automata, which preserves exact values (and not only boundedness properties as it is usually the case in the framework of cost functions). We then showed that the expressive power of LTL$^{\leq}$ in terms of cost functions is the same as aperiodic stabilization semigroups. The proof uses a new syntactic congruence, which has a general interest in the study of regular cost functions. This result implies the decidability of the LTL$^{\leq}$-definable class of cost functions.

As a further work, we can try to put $\omega\sharp$-expressions in a larger framework, by doing an axiomatization of $\omega\sharp$-semigroups. We can also extend this work to infinite words, and define an analog to Büchi automata for cost functions. To continue the analogy with classic languages results, we can define a quantitative extension of FO describing the same class as LTL$^{\leq}$, and search for analog definitions of counter-free B-automata and star-free B-regular expressions. The translation from LTL$^{\leq}$-formulae to B-automata can be further studied in terms of optimality of number of counters of the resulting B-automaton.

Acknowledgments

I am very grateful to my advisor Thomas Colcombet for our helpful discussions, and for the guidelines he gave me on this work, and to Michael Vanden Boom for helping me with language and presentation issues.

References

1 Mikolaj Bojańczyk and Thomas Colcombet. Bounds in ω-regularity. In *LICS 06*, pages 285–296, August 2006.

2 Thomas Colcombet. The theory of stabilization monoids and regular cost functions. *ICALP, Lecture Notes in Computer Science*, 2009.

3 Thomas Colcombet, Denis Kuperberg, and Sylvain Lombardy. Regular temporal cost functions. In *ICALP (2)*, pages 563–574, 2010.

4 Stéphane Demri and Paul Gastin. Specification and verification using temporal logics. In *Modern applications of automata theory*, volume 2 of *IISc Research Monographs*. World Scientific, 2010. To appear.

5 Kosaburo Hashiguchi. Relative star height, star height and finite automata with distance functions. In *Formal Properties of Finite Automata and Applications*, pages 74–88, 1988.

6 Kosaburo Hashiguchi. Improved limitedness theorems on finite automata with distance functions. *Theor. Comput. Sci.*, 72(1):27–38, 1990.

7 Daniel Kirsten. Distance desert automata and the star height problem. *RAIRO*, 3(39):455–509, 2005.

8 Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. From liveness to promptness. *Formal Methods in System Design*, 34(2):83–103, 2009.

9 M.-P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control 8*, pages 190–194, 1965.

10 Moshe Y. Vardi and Pierre Wolper. Automata-theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci.*, 32(2):183–221, 1986.

11 Thomas Wilke. Classifying discrete temporal properties. In Christoph Meinel and Sophie Tison, editors, *STACS*, volume 1563 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1999.

Bounds on the maximum multiplicity of some common geometric graphs*

Adrian Dumitrescu[1], André Schulz[2], Adam Sheffer[3], and Csaba D. Tóth[4]

1 **Department of Computer Science, University of Wisconsin–Milwaukee, USA**
dumitres@uwm.edu

2 **Institut für Mathematsche Logik und Grundlagenforschung, Universität Münster, Germany**
andre.schulz@uni-muenster.de

3 **School of Computer Science, Tel Aviv University, Israel**
sheffera@tau.ac.il

4 **Department of Mathematics and Statistics, University of Calgary, Canada**
cdtoth@ucalgary.ca

Abstract

We obtain new lower and upper bounds for the maximum multiplicity of some weighted, and respectively non-weighted, common geometric graphs drawn on n points in the plane in general position (with no three points collinear): perfect matchings, spanning trees, spanning cycles (tours), and triangulations.

(i) We present a new lower bound construction for the maximum number of triangulations a set of n points in general position can have. In particular, we show that a *generalized double chain* formed by two *almost convex* chains admits $\Omega(8.65^n)$ different triangulations. This improves the bound $\Omega(8.48^n)$ achieved by the previous best construction, the *double zig-zag chain* studied by Aichholzer *et al.*

(ii) We present a new lower bound of $\Omega(11.97^n)$ for the number of *non-crossing* spanning trees of the *double chain* composed of two *convex* chains. The previous bound, $\Omega(10.42^n)$, stood unchanged for more than 10 years.

(iii) Using a recent upper bound of 30^n for the number of triangulations, due to Sharir and Sheffer, we show that n points in the plane in general position admit at most $O(68.664^n)$ non-crossing spanning cycles.

(iv) We derive exponential lower bounds for the number of maximum and minimum *weighted* geometric graphs (matchings, spanning trees, and tours). It was known that the number of longest non-crossing spanning trees of a point set can be exponentially large, and here we show that this can be also realized with points in convex position. For points in convex position we obtain tight bounds for the number of longest and shortest tours. We give a combinatorial characterization of the longest tours, which leads to an $O(n \log n)$ time algorithm for computing them.

1998 ACM Subject Classification G.2.1 Counting problems

Keywords and phrases Combinatorial geometry, matching, triangulation, spanning tree, spanning cycle, weighted structure, non-crossing condition.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.637

* Adrian Dumitrescu was supported in part by NSF CAREER grant CCF-0444188. André Schulz was supported by the German Research Foundation (DFG) under grant SCHU 2458/1-1. Adam Sheffer was supported by Grant 338/09 from the Israel Science Fund. Csaba Tóth was supported in part by NSERC grant RGPIN 35586. Research by Tóth was conducted at Tufts University.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 637–648
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

SYMPOSIUM
ON THEORETICAL
ASPECTS
OF COMPUTER
SCIENCE

1 Introduction

Let P be a set of n points in the plane in *general position*, i.e., no three points lie on a common line. A *geometric graph* $G = (P, E)$ is a graph drawn in the plane so that the vertex set consists of the points in P and the edges are drawn as straight line segments between points in P. All graphs we consider in this paper are geometric graphs. We call a graph *non-crossing* if edges intersect only at common endpoints.

It is a fundamental question to determine the maximum number of non-crossing geometric graphs on n points in the plane. We follow common conventions (see e.g., [19]) and denote by $\mathtt{pg}(P)$ the number of non-crossing (plane) graphs on P, and by $\mathtt{pg}(n) = \max_{|P|=n} \mathtt{pg}(P)$ the *maximum number* of non-crossing graphs an n-element point set can admit. Analogously, we introduce shorthand notation for the maximum number of triangulations, perfect matchings, spanning trees, and spanning cycles (i.e., Hamiltonian cycles); see Table 1.

Abbr.	Graph class	Lower bound	Upper bound
$\mathtt{pg}$(n)	graphs	$\Omega(41.18^n)$ [1, 13]	$O(207.85^n)$ [14, 21]
$\mathtt{cf}$(n)	cycle-free graphs	$\mathbf{\Omega(12.23^n)}$ [new, Thm. 2]	$O(164.49^n)$ [14, 21]
$\mathtt{pm}$(n)	perfect matchings	$\Omega^*(3^n)$ [13]	$O(10.07^n)$ [19]
$\mathtt{st}$(n)	spanning trees	$\mathbf{\Omega(11.97^n)}$ [new, Thm. 2]	$O(146.37^n)$ [14, 21]
$\mathtt{sc}$(n)	spanning cycles	$\Omega(4.64^n)$ [13]	$\mathbf{O(68.664^n)}$ [new, Thm. 3]
$\mathtt{tr}$(n)	triangulations	$\mathbf{\Omega(8.65^n)}$ [new, Thm. 1]	$O(30^n)$ [21]

Table 1 Classes of non-crossing geometric graphs, current best upper and lower bounds.

In the past 30 years numerous researchers have tried to estimate these quantities. In a pivotal result, Ajtai *et al.* [2] showed that $\mathtt{pg}(n) = O(c^n)$ for an absolute, but very large constant $c > 0$. The constant c has been improved several times since then, the best bound today is $c < 207.85$, which follows form the combination of the result of Sharir and Sheffer [21] with the result of Hoffmann *et al.* [14]. Interestingly, this upper bound, as well as the currently best upper bounds for $\mathtt{st}(n)$, $\mathtt{sc}(n)$, and $\mathtt{cf}(n)$, are derived from upper bounds on $\mathtt{tr}(n)$. This underlines the importance of the bound for $\mathtt{tr}(n)$ in this setting. For example, the best known upper bound for $\mathtt{st}(n)$ is the combination of $\mathtt{tr}(n) \leq 30^n$ [21] with the ratio $\mathtt{sc}(n)/\mathtt{tr}(n) = O^*(4.879^n)$ [14]; see also previous work [17, 18, 19, 20]. To our knowledge, the only upper bound derived via a different approach is for the number of perfect matchings by Sharir and Welzl [19], $\mathtt{pm}(n) = O(10.07^n)$.

So far, we recalled various upper bounds on the maximum number of geometric graphs in certain classes. In this paper we mostly conduct our offensive from the other direction, on improving the corresponding *lower bounds.* Lower bounds for unweighted non-crossing graph classes were obtained in [1, 7, 13]. García, Noy, and Tejel [13] were the first to recognize the power of the *double chain* configuration in establishing good lower bounds for the maximum number of matchings, triangulations, spanning cycles and trees. It was widely believed for some time that the double chain gives asymptotically the highest number of triangulations, namely $\Theta^*(8^n)$. This was until 2006, when Aichholzer *et al.* [1] showed that another configuration, the so-called *double zig-zag chain*, admits $\Theta^*(\sqrt{72}^n) = \Omega(8.48^n)$ triangulations[1]. In this paper we further exploit the power of *almost convex* polygons and establish a new lower bound $\mathtt{tr}(n) = \Omega(8.65^n)$. For matchings, spanning cycles, and plane graphs, the double chain still holds the current record.

[1] We use the Θ^*, O^*, Ω^* notation for the asymptotic growth of functions ignoring polynomial factors.

Less studied are multiplicities of *weighted* geometric graphs. The weight of a geometric graph is the sum of its (Euclidean) edge lengths. This leads to the question how many graphs of a certain type (e.g., matchings, spanning trees, or tours) with *minimum* or *maximum* weight can be realized on an n-element point set. The notation is analogous; see Table 2. Dumitrescu [8] showed that the longest and shortest matchings can have exponential multiplicity, $2^{\Omega(n)}$, for a point set in general position. Furthermore, the longest and shortest spanning trees can also have multiplicity of $2^{\Omega(n)}$. Both bounds count explicitly geometric graphs with crossings; however these minima are automatically non-crossing. The question for the maximum multiplicity for non-crossing geometric graphs remained open for most of the geometric graph classes. Since we do not have any upper bounds that are better than those for the corresponding unweighted classes, the "upper bound" column is missing from Table 2.

Abbr.	**Graph class**	**Lower bound**
$\mathtt{pm}_{\min}(n)$	shortest perfect matchings	$\Omega(2^{n/4})$ [8]
$\mathtt{pm}_{\max}(n)$	longest perfect matchings	$\boldsymbol{\Omega(2^{n/4})}$ [new, Theorem 4]
$\mathtt{st}_{\min}(n)$	shortest spanning trees	$\Omega(2^{n/2})$ [8]
$\mathtt{st}_{\max}(n)$	longest spanning trees	$\boldsymbol{\Omega(2^{n})}$ [new, Theorem 7]
$\mathtt{sc}_{\min}(n)$	shortest spanning cycles	$\boldsymbol{\Omega(2^{n/3})}$ [new, Theorem 8]
$\mathtt{sc}_{\max}(n)$	longest spanning cycles	$\boldsymbol{\Omega(2^{n/3})}$ [new, Theorem 5]

Table 2 Classes of *weighted* non-crossing geometric graphs: exponential lower bounds.

Our results. Due to space constraints, some of the proofs are omitted from this extended abstract (all proofs are available in the full version of this paper [10]).

(I) A new lower bound, $\Omega(8.65^n)$, for the maximum number of triangulations a set of n points can have. We first re-derive the bound given by Aichholzer *et al.* [1] with a simpler analysis, which allows us to extend it to more complex point sets. Our estimate might be the best possible for the type of construction we consider.

(II) A new lower bound, $\Omega(11.97^n)$, for the maximum number of non-crossing spanning trees a set of n points can have. This is obtained by refining the analysis of the number of such trees on the "double chain" point configuration. The previous bound was $\Omega(10.42^n)$. A slight modification of the construction improves also the lower bound for cycle-free non-crossing graphs. In particular, we improve the old bound of $\Omega(11.62^n)$ to $\Omega(12.23^n)$,

(III) A new upper bound, $O(68.664^n)$, for the number of non-crossing spanning cycles on n points in the plane. This improves the latest upper bound of 70.21^n obtained by a combination of the results of Buchin *et al.* [4] and a recent upper bound of 30^n on the number of triangulations by Sharir and Sheffer [21].

(IV) Bounds on the maximum multiplicity of various weighted geometric graphs (weighted by Euclidean length). We show that the maximum number of longest non-crossing perfect matchings, spanning trees, spanning cycles, as well as shortest tours are all exponential in n. We also derive tight bounds, as well as a combinatorial characterization of longest tours over points in convex position. This yields an $O(n \log n)$ algorithm to compute a longest tour for such sets.

1.1 Preliminaries

Asymptotics of multinomial coefficients. Denote by $H(q) = -q \log q - (1-q)\log(1-q)$ the *binary entropy function*, where log stands for the logarithm in base 2 (by convention,

$0 \log 0 = 0$). For a constant $0 \le \alpha \le 1$, the following estimate can be easily derived from Stirling's formula for the factorial:

$$\binom{n}{\alpha n} = \Theta(n^{-1/2} 2^{H(\alpha)n}), \tag{1}$$

We also need the following bound on the sum of binomial coefficients; see [3] for a proof and [9, 11] for an application. If $0 < \alpha \le \frac{1}{2}$ is a constant,

$$\sum_{k=0}^{k \le \alpha n} \binom{n}{k} \le 2^{H(\alpha)n}. \tag{2}$$

Define similarly the *generalized entropy function* of k parameters $\alpha_1, \ldots, \alpha_k$, satisfying

$$\sum_{i=1}^{k} \alpha_i = 1, \quad \alpha_1, \ldots, \alpha_k \ge 0, \text{ as } \; H_k(\alpha_1, \ldots, \alpha_k) = -\sum_{i=1}^{k} \alpha_i \log \alpha_i. \tag{3}$$

Clearly, $H(q) = H_2(q, 1-q)$. Recall, the multinomial coefficient

$$\binom{n}{n_1, n_2, \ldots, n_k} = \frac{n!}{n_1! n_2! \ldots n_k!},$$

where $\sum_{i=1}^{k} n_i = n$, counts the number of distinct ways to permute a multiset of n elements, k of which are distinct, with n_i, $i = 1, \ldots, k$, being the multiplicities of each of the k distinct elements.

Assuming that $n_i = \alpha_i n$, $i = 1, \ldots, k$, for constants $\alpha_1, \ldots, \alpha_k$, satisfying (3), again by using Stirling's formula for the factorial, one gets an expression analogous to (1):

$$\binom{n}{n_1, n_2, \ldots, n_k} = \Theta(n^{-(k-1)/2}) \cdot \left(\prod_{i=1}^{k} \alpha_i^{-\alpha_i} \right)^n = \Theta(n^{-(k-1)/2}) \cdot 2^{H_k(\alpha_1, \ldots, \alpha_k)n}. \tag{4}$$

Notations and conventions. For a polygonal chain P, let $|P|$ denote the number of vertices. If $1 < c_1 < c_2$ are two constants, we frequently write $\Omega^*(c_2^n) = \Omega(c_1^n)$. We also write $f(n) \sim g(n)$ whenever $f(n) = \Theta(g(n))$.

2 Lower bound on the maximum number of triangulations

Following the notation from [15], we denote by $P(n, k^r)$ the class of *almost convex polygons* with n vertices, formed by concatenating r flat reflex chains, each having k interior vertices. For example, $P(n, 0^r)$ is the class of convex polygons with $n = r$ vertices. Note that, for $r \ge 3$ and $k \ge 0$, every polygon in $P(n, k^r)$ has $n = r(k+1)$ vertices, r of which are convex. See Fig. 1 for a small example. To further simplify notation, we denote by $P(n, k^r)$ any polygon in this class; note they are all equivalent in the sense that they have the same visibility graph.

In establishing our new bound on the maximum number of triangulations, we go through the following steps: We first describe the double zig-zag chain from [1] in our framework, and re-derive the $\Theta^*(\sqrt{72}^n)$ bound of [1] for the number of its triangulations. Our simpler analysis extends to some variations of the double zig-zag chains, and leads to a new lower bound of $\texttt{tr}(n) = \Omega(8.65^n)$.

Two x-monotone polygonal chains L and U are said to be *mutually visible* if every pair of points $p \in L$ and $q \in U$, are *visible* from each other. Let us call $D(n, k^r)$ the *generalized double*

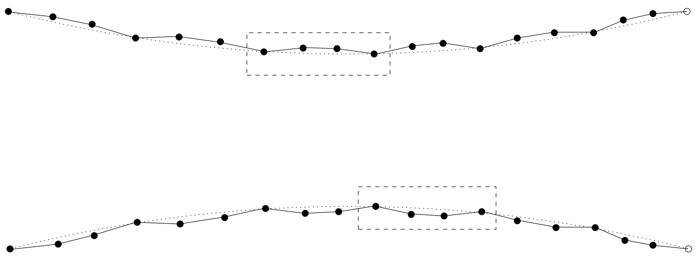


Figure 1 Two (flat) mutually visible copies of $P(18, 2^6)$ that form $D(36, 2^{12})$. Two consecutive hull vertices of $P(18, 2^6)$ with a reflex chain of two vertices in between are indicated in both the upper and the lower chain.

chain of n points made up of the set of points in two mutually visible copies of $P(n/2, k^r)$, each with $n/2 = r(k+1)$ vertices, with opposite concavities as in Fig. 1[2]. Generalized double chains are a family of point configurations, containing, among others, the double chain and double zig-zag chain configurations. In particular, $D(n, 1^r)$ is the *double zig-zag chain* used by Aichholzer *et al.* [1].

▶ **Theorem 1.** *The point set $D(n, 3^r)$ with $n = 8r$ points admits $\Omega(8.65^n)$ triangulations.*

Proof. We start by estimating the number of triangulations of $P(n, k^r)$. Denote this number by $t(n, k^r) = \texttt{tr}(P(n, k^r))$. Recall that $P(n, k^r)$ has $n = r(k + 1)$ vertices. According to [15, Theorem 3],

$$t(n, k^r) \sim \left(\frac{1 + k/2}{2^k}\right)^r \cdot t(n) \sim \left(\frac{k+2}{2^{k+1}}\right)^r \cdot 4^{r(k+1)} = \left((k+2)^{\frac{1}{k+1}} \cdot 2\right)^n .$$

In particular,

- for $k = 1$, $t(n, 1^r) \sim (2\sqrt{3})^n = \sqrt{12}^n$. This estimate was used for counting triangulations in the construction $D(n, 1^r)$ with $\Omega(8.48^n)$ triangulations from [1].
- for $k = 2$, $t(n, 2^r) \sim (2^{5/3})^n$.
- for $k = 3$, $t(n, 3^r) \sim (5^{1/4} \cdot 2)^n$.
- for $k = 4$, $t(n, 4^r) \sim (6^{1/5} \cdot 2)^n$.

The following estimate is used in all our triangulation bounds. Consider two mutually visible polygonal chains, L and U, with m vertices each (L is the lower chain and U is the upper chain). As in the proof of [13, Theorem 4.1], the region between the two chains consists of $2m - 2$ triangles, such that exactly $m - 1$ triangles have an edge along L and the remaining $m - 1$ triangles have an edge adjacent to U. It follows that the number of distinct triangulations of this middle region is

$$\binom{2m-2}{m-1} = \Theta(m^{-1/2} \cdot 4^m). \tag{5}$$

The old $\Omega(8.48^n)$ lower bound in a new perspective. We estimate from below the number of triangulations of $D(n, 1^r)$ as follows. Recall that $|L| = |U| = n/2 = 2r$. Include all edges of L and U in any of the triangulations we construct. Now construct different triangulations as follows. Independently select a subset of $\alpha_1 r$ short edges of $\mathrm{conv}(U)$ and

[2] For convenience, an extra vertex is added to each chain to complete the last group in the figure.

similarly, a subset of $\alpha_1 r$ short edges of $\mathrm{conv}(L)$. Here $\alpha_1 \in (0,1)$ is a constant to be determined later. According to (1), this can be done in

$$\binom{r}{\alpha_1 r} = \Theta(r^{-1/2} \cdot 2^{H(\alpha_1)r})$$

ways in each of the two chains. Include these edges in the triangulation. Observe that after adding these short edges the middle region between the (initial) chains L and U is sandwiched between two mutually visible shorter chains, say $L' \subset L$ and $U' \subset U$, where

$$|L'| = |U'| = 2r - \alpha_1 r = (2-\alpha_1)r. \tag{6}$$

Triangulate this middle region in all possible ways, as outlined in the paragraph above (5). Let N denote the total number of triangulations of $D(n,1^r)$ obtained in this way. By the above estimate, we have $t(n,1^r) \sim (2\sqrt{3})^n$. Combining this with (5) and (6),

$$\begin{aligned} N &= \Omega^*\left(\left[(2\sqrt{3})^{2r} 2^{H(\alpha_1)r}\right]^2 4^{(2-\alpha_1)r}\right) = \Omega^*\left(\left[2^{2r}3^r 2^{(2-\alpha_1)r} 2^{H(\alpha_1)r}\right]^2\right) = \\ &= \Omega^*\left(\left[2^2 \cdot 3 \cdot 2^{(2-\alpha_1)} 2^{H(\alpha_1)}\right]^{2r}\right) = \Omega^*\left(\left[2^{4-\alpha_1+H(\alpha_1)} \cdot 3\right]^{n/2}\right) = \Omega^*\left(a^n\right), \end{aligned}$$

where

$$a = \left[2^{4-\alpha_1+H(\alpha_1)} \cdot 3\right]^{(1/2)}.$$

By setting $\alpha_1 = 1/3$, as in [1], this yields $a = 6\sqrt{2} = 8.485\ldots$, and $N = \Omega^*(8.485^n) = \Omega(8.48^n)$.

Applying a similar analysis for a generalized double chain with reflex chains of length 3 implies Theorem 1. The details are in the full paper [10]. ◀

3 Lower bound on the maximum number of non-crossing spanning trees and forests

In this section we derive a new lower bound for the number of non-crossing spanning trees on the double-chain $D(n,0^r)$, hence also for the maximum number of non-crossing spanning trees an n-element planar point set can have. The previous best bound, $\Omega(10.42^n)$, is due to Dumitrescu [8]. By refining the analysis of [8] we obtain a new bound $\Omega(11.97^n)$.

▶ **Theorem 2.** *For the double chain $D(n,0^r)$, we have*

$$\Omega(11.97^n) < \mathtt{st}(D(n,0^r)) < O(24.68^n), \text{ and}$$
$$\Omega(12.23^n) < \mathtt{cf}(D(n,0^r)) < O(24.68^n).$$

These bounds imply that $\mathtt{st}(n) = \Omega(11.97^n)$ *and* $\mathtt{cf}(n) = \Omega(12.23^n)$.

Instead of spanning trees, we count (spanning) forests formed by two trees, similarly to [8]. One of the trees will be associated with the lower chain L and is called *lower tree*, the other tree will be associated with the upper chain U and is called *upper tree*. Since the two trees can be connected in at most $O(n^2)$ ways, it is enough the bound the number of two trees. Fig. 2 shows an example. We count only special kinds of forests: no edge of the lower tree connects two vertices of the upper chain, and similarly, no edge of the upper tree connects two vertices of the lower chain. We call the connected components of the edges between U and L *bridges*. For the class of forests we consider, bridges are subtrees of the

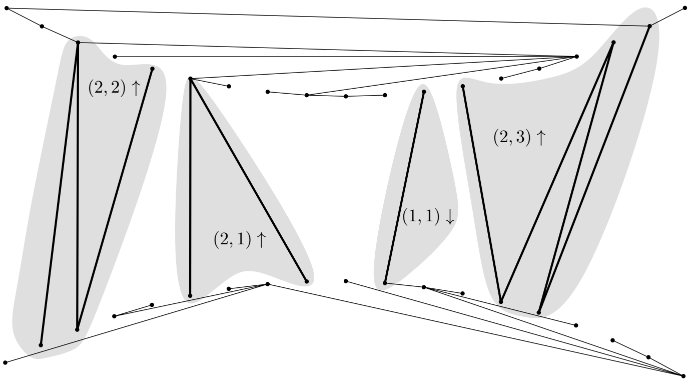


Figure 2 A double chain with lower and upper tree and four bridges.

lower or the upper tree. A bridge is called an (i, j)-*bridge* if it has i vertices in L and j vertices in U. Every bridge is part of either the upper or the lower tree. We say that in the first case the bridge is oriented *upwards* and in the latter case it is oriented *downwards*. Since edges cannot cross, the bridges have a natural left-to-right order. Fig. 2 shows four bridges, the first bridge is an upward oriented $(2, 2)$-bridge. We consider only bridges (i, j), with $1 \le i, j \le z$, for some fixed positive integer z. For $z = 1$, our analysis coincides with the one in [8], and we rederive the lower bound of $\Omega(10.42^n)$ found there. Successive improvements will be achieved by considering $z = 2, 3, 4$.

Let $m = n/2$ be the number of points on one chain. The distribution of bridges is specified by a set of parameters α_{ij}, to be determined later, where the number of (i, j)-bridges is $\alpha_{ij}m$. To simplify further expressions we introduce the following wildcard-notation:

$$\alpha_{i*} = \sum_{k=1}^{z} \alpha_{ik}, \quad \alpha_{*j} = \sum_{k=1}^{z} \alpha_{kj}, \quad \text{and} \quad \alpha_{**} = \sum_{k=1}^{z} \alpha_{*k} = \sum_{k=1}^{z} \alpha_{k*}.$$

A vertex is called a *bridge vertex*, if it is part of some bridge, and it is a *tree vertex* otherwise. We denote by $\alpha_L m$ the number of bridge vertices along L, and by $\alpha_U m$ the number of bridge vertices along U, we have

$$\alpha_L = \sum_{k=1}^{z} k\alpha_{k*}, \quad \text{and} \quad \alpha_U = \sum_{k=1}^{z} k\alpha_{*k}.$$

To count the forests we proceed as follows. We first count the distributions of the vertices that belong to bridges on the lower (N_L) and upper chain (N_U). We then count the different ways how bridges can be realized (N_{bridges}) and how the bridges can be connected to the two trees (N_{links}). Finally, we estimate the number of the trees within the two chains (N_{trees}). All these numbers are parameterized by the variables α_{ij}.

Consider the feasible locations of bridge vertices at the lower chain. We have $\binom{m}{\alpha_L m}$ choices to select the bridge vertices in L. Every bridge vertex belongs to some (i, j)-bridge. The vertices of the bridges cannot interleave, thus we can describe the configuration of bridges by a sequence of (i, j) tuples that denotes the appearance of the $\alpha_{**}m$ bridges from left to right on L. There are $\binom{\alpha_{**}m}{\alpha_{11}m, \alpha_{12}m, \ldots, \alpha_{zz}m}$ such sequences. This give us a total of

$$N_L := \binom{m}{\alpha_L m} \binom{\alpha_{**}m}{\alpha_{11}m, \alpha_{12}m, \ldots, \alpha_{zz}m} = \Theta^*\left(2^{H(\alpha_L)m + \alpha_{**}H_{(z^2)}(\alpha_{11}/\alpha_{**}, \ldots, \alpha_{zz}/\alpha_{**})m}\right)$$

such “configurations” of bridge vertices along L.

We now determine how many options we have to place the bridge vertices on U. Since we have already specified the sequence of the (i,j)-bridges at the lower chain, all we can do is to select the bridge vertices in U. This gives

$$N_U := \binom{m}{\alpha_U m} = \Theta^*\left(2^{H(\alpha_U)m}\right)$$

possibilities for the configuration on U.

We now study in how many ways the bridges can be added to the two trees. Since all bridges are subtrees, we can link one of the bridge vertices with the lower or upper tree. From this perspective the whole bridge acts like a super-node in one of the trees. The orientation of the bridges determine which tree they are glued to: upwards bridges to the upper tree, downwards bridges to the lower tree. For every pair (i,j) we orient half of the (i,j)-bridges upwards and half of them downwards. To glue the bridges to the trees we have to specify a vertex that will be linked to one of the trees. Depending on the orientation of the (i,j)-bridge, we have i candidates for a downwards oriented bridge and j candidates for an upward oriented bridge. In total we have

$$N_{\text{links}} := \prod_{i,j} \binom{\alpha_{i,j}m}{\alpha_{i,j}m/2} \left(i^{\alpha_{ij}/2} j^{\alpha_{ij}/2}\right)^m = \prod_{i,j} \Theta^*\left(2^{\alpha_{i,j}m}\right)(ij)^{\frac{\alpha_{ij}m}{2}} = \Theta^*\left(2^{\alpha_{**}m}\right) \prod_{i,j} (ij)^{\frac{\alpha_{ij}m}{2}}$$

ways to link the bridges with the trees.

Until now we have specified which vertices belong to which type of bridges, the orientation of the bridges, and the vertex where the bridge will be linked to its tree. It remains to count the number of ways to actually "draw" the bridges. Let us consider an (i,j)-bridge. All edges have to go from L to U and the bridge has to be a tree. The number of such trees equals the number of triangulations of a polygon with point set $\{(k,0) \mid 0 \le k \le i\} \cup \{(k,1) \mid 0 \le k \le j\}$. By deleting the edges along the horizontal lines $y = 0$ and $y = 1$, we define a bijection between these triangulations and the combinatorial types of (i,j)-bridges. The number of triangulations is now easy to express similarly to Equation (5): We have $i+j-2$ triangles, and each triangle is adjacent to a horizontal edge along either $y = 0$ or $y = 1$, where exactly $i-1$ triangles are adjacent to line $y = 0$. In total we have $B_{ij} := \binom{i+j-2}{i-1}$ different triangulations and therefore we can express the number of different bridges by

$$N_{\text{bridges}} = \left(\prod_{ij} B_{ij}^{\alpha_{ij}}\right)^m.$$

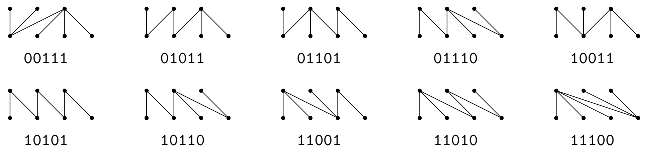


Figure 3 All $B_{34} = 10$ combinatorial types of $(3,4)$-bridges. If an edge differs form its predecessor at the top we write a `0`, otherwise a `1`. We obtain a bijection between the bridges and sequences with three `1`s and two `0`s.

Observe that the upper and the lower trees are trees on a convex point set. By considering the bridges as super-nodes, we treat the lower chain as a convex chain of n_L vertices. Similarly, we think of the upper chain as a convex chain with n_U vertices. We have

$$n_U = \left(1 - \sum_{k=1}^{n} \frac{2k-1}{2}\alpha_{k*}\right)m, \qquad \text{and} \qquad n_L = \left(1 - \sum_{k=1}^{n} \frac{2k-1}{2}\alpha_{*k}\right)m.$$

(Notice that the bridges take away all of its vertices, except one, depending on the orientation). Since the number of non-crossing spanning trees on an n-element convex point set equals $\Theta^*((27/4)^n)$ [12], the number of spanning trees within the two chains is given by

$$N_{\text{trees}} = O^*\Big(\big(27/4\big)^{n_L+n_U}\Big).$$

To finish our analysis we have to find the optimal parameters α_{ij} such that

$$\texttt{st}(D(n,0^r)) = \Omega^*\left(N_L \cdot N_U \cdot N_{\text{bridges}} \cdot N_{\text{links}} \cdot N_{\text{trees}}\right) \tag{7}$$

is maximized. The details are presented in the full paper [10].

4 Upper bound for the number of non-crossing spanning cycles

Newborn and Moser [16] asked what is the maximum number of non-crossing spanning cycles for n points in the plane, and they proved $\Omega((10^{1/3})^n) \le \texttt{sc}(n) \le O(6^n\lfloor\frac{n}{2}\rfloor!)$. The first exponential upper bound $\texttt{sc}(n) \le 10^{13n}$ was obtained by Ajtai *et al.* [2], and has been followed by a series of improved bounds (e.g., see [4, 7, 19], a more comprehensive history can be found in [6]). Currently, the best known lower bound $4.462^n \le \texttt{sc}(n)$ is by García *et al.* [13]. The previous best upper bound $O(70.21^n)$ is obtained by combining the upper bound $30^{n/4}$ of Buchin *et al.* [4] for the number of spanning cycles in a triangulation with a new upper bound of $\texttt{tr}(n) \le 30^n$ by Sharir and Sheffer [21].

The bound by Buchin *et al.* [4] cannot be improved much further, since they also present triangulations with $\Omega(2.0845^n)$ spanning cycles. However, the bound for $\texttt{sc}(n)$ still seems rather weak since it potentially counts some spanning cycles many times. To overcome this inefficiency, we use the notion of *pseudo-simultaneously flippable edges* (*ps-flippable edges* for short), introduced in [14]. A set F of edges in a triangulation is ps-flippable if after deleting all edges in F, the bounded faces are convex. One can obtain a lower bound for the support of a spanning cycle C in terms of the number of ps-flippable edges that are *not* in C.

▶ **Theorem 3.** *We have* $\texttt{sc}(n) = O\left(68.664^n\right)$.

The proof is available in the full paper [10].

5 Weighted geometric graphs

Longest perfect matchings. Let n be even, and consider perfect matchings on a set of n points in the plane. It is easy to construct n-element point sets (no three of which are collinear) with an exponential number of *longest* matchings: [8] gives constructions with $\Omega(2^{n/4})$ such matchings. Moreover, the same lower bound can be achieved with yet another restriction, convex position, imposed on the point set; see [8]. Here, we present constructions with an exponential number of *maximum* (longest) non-crossing matchings.

▶ **Theorem 4.** *For every even n, there exist n-element point sets with at least $2^{\lfloor n/4\rfloor}$ longest non-crossing perfect matchings. Consequently,* $\texttt{pm}_{\max}(n) = \Omega(2^{n/4})$.

Proof. (sketch) Assume first that n is a multiple of 4. Let $S_4 = \{a, b, c, d\}$ be a 4-element point set such that segment ab is vertical, cd lies on the orthogonal bisector of ab (hence, $|ac| = |bc|$ and $|ad| = |bd|$), $|ab| = |cd| = \frac{1}{n}$ and $\min\{|ac|, |ad|\} = |ac| = |bc| = 2n$. Then S_4 has two maximum matchings, $\{ac, bd\}$ and $\{ad, bc\}$, each of which has length at least $4n$. Let the n-element point set P be the union of $n/4$ translated copies of S_4 lying in disjoint horizontal strips such that the copies of a are almost collinear, all the copies of points a and b lie in a disk of unit diameter, and all the copies of points c and d lie in a disk of unit diameter; see Fig. 4.

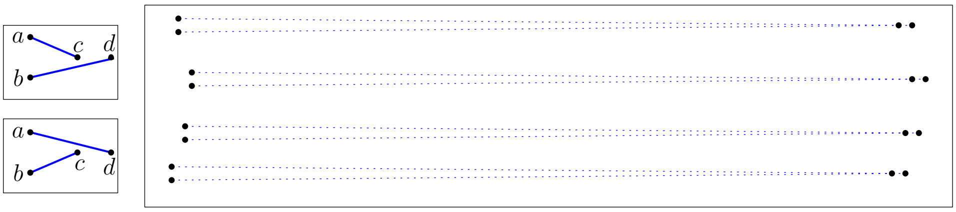


Figure 4 Left: Two possible maximum matchings for the point set $S_4 = \{a, b, c, d\}$. Right: A set of $n = 16$ points that admit 2^4 maximum non-crossing perfect matchings.

If we combine the maximum matchings of all copies of S_4, then we obtain $2^{n/4}$ non-crossing perfect matchings of P. All these matchings have the same length, which is at least $\frac{n}{4} \cdot 4n = n^2$. In the full paper [10], we show that this is the *maximum* possible length of a non-crossing perfect matching of P. ◀

Longest non-crossing tours. By Theorem 8, the maximum number of shortest non-crossing spanning cycles on n points is exponential in n. We show here that the maximum number of longest non-crossing spanning cycles is also exponential in n.

▶ **Theorem 5.** *Let $\mathtt{sc}_{\max}(n)$ denote the maximum number of longest non-crossing spanning cycles that an n-element point set can have. Then we have $\mathtt{sc}_{\max}(n) = \Omega(2^{n/3})$.*

Proof. (sketch) For every $k \in \mathbb{N}$, we construct a set Q of $4k + 1$ points that admits $2^k = \Omega(2^{n/4})$ longest non-crossing tours. We start by constructing an auxiliary set P of $2k$ points. The auxiliary point set P may contain collinear triples, however our final set Q does not. Recall that two segments cross if and only if their relative interiors intersect. We construct $P = \{c_i, x_i : i = 1, 2, \ldots, k\}$ with the following properties: (i) for every x_i, the farthest point in P is c_i; (ii) the perfect matching $M = \{c_i x_i : i = 1, 2, \ldots, k\}$ is non-crossing; and (iii) the convex hull of P is $\mathrm{conv}(P) = (x_1, c_1, c_2, \ldots, c_k)$. Note that property (i) implies that M is the maximum matching of P.

For $k \in \mathbb{N}$, let $\alpha = \frac{\pi}{3k}$. We construct $P = \{c_i, x_i : i = 1, \ldots, k\}$ iteratively. During the iterative process, we maintain the properties that $|x_i c_i| > \max_{j<i} |x_i c_j|$ and $|x_{i+1} c_i| > \max_{j<i} |x_{i+1} c_j|$. Initially, let $c_1 = (0, 0)$, $x_1 = (2, 0)$, and $x_2 = (2 - \frac{1}{k}, 0)$. Let $\vec{\ell}_1$ be a ray emitted by x_1 and incident to c_1. Refer to Fig. 5. If c_i, x_i and x_{i+1} are already defined, we construct points c_{i+1} and x_{i+2} (in the last iteration, only c_{i+1}) as follows. Let $\vec{\ell}_{i+1}$ be a ray emitted by x_{i+1} such that $\angle(\vec{\ell}_{i+1}, \vec{\ell}_i) = \alpha$. Compute the intersections of ray $\vec{\ell}_{i+1}$ with the circle centered at x_i of radius $|x_i c_i|$ and the circle centered at x_{i+1} of radius $|x_{i+1} c_i|$. Let $c_{i+1} \in \vec{\ell}_{i+1}$ be the midpoint of the segment between these two intersection points. This choice guarantees that $|x_{i+1} c_{i+1}| > |x_{i+1} c_j|$ and $|x_j c_{i+1}| < |x_j c_j|$ for all $j \leq i$. Now let $x_{i+2} \in c_{i+1} x_{i+1}$ be a point at distance at most $\frac{1}{k}$ from x_{i+1} such that we have $|x_{i+2} c_{i+1}| > |x_{i+2} c_j|$ for all $j \leq i$. This completes the description of P.

Note that $|x_i x_{i+1}| \leq \frac{1}{k}$, and so the points $x_1, \ldots, x_k$ lie in a disk of diameter 1. Hence, for every point x_i, the farthest point in P is in $\{c_j : j = 1, \ldots, k\}$. By the above construction, the farthest point from x_i in $\{c_j : j = 1, \ldots, k\}$ is c_i. This proves that P has property (i). It is easy to verify that P has properties (ii) and (iii), as well.

We now construct the point set Q based on P. Let $\delta > 0$ be a sufficiently small constant. For every segment $c_i x_i$ we construct a skinny deltoid $\Delta_i = (a_i, b_i, c_i, d_i)$, see

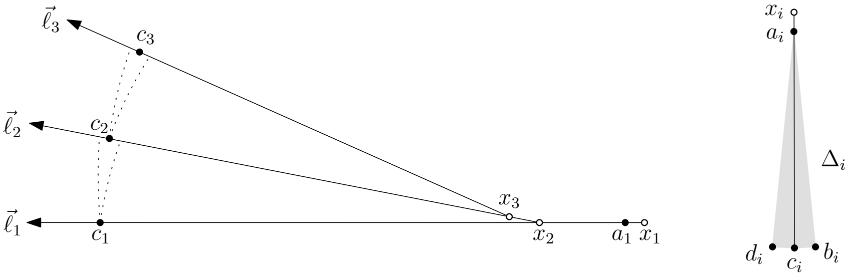


Figure 5 Left: The auxiliary point set P for $k = 3$. Right: A long and skinny deltoid $\Delta_i = (a_i, b_i, c_i, d_i)$.

Fig. 5, such that $a_i \in c_i x_i$ is at distance δ from x_i, we have $|b_i c_i| = |c_i d_i| = \delta$, and $|a_i b_i| = |a_i c_i| = |a_i d_i| = |c_i x_i| - \delta$. Since the segments $c_i x_i$ are pairwise non-crossing and $\delta > 0$ is small, the deltoids Δ_i are pairwise interior disjoint. Let Q be the set of vertices of all deltoids Δ_i, $i = 1, \ldots, k$, and the point x_1. Since $\text{conv}(P) = (c_1, c_2, \ldots, c_k, x_1)$, we have $\text{conv}(Q) = (b_1, c_1, d_1, b_2, c_2, d_2, \ldots, b_k, c_k, d_k, x_1)$, and the points $\{a_i : i = 1, \ldots, k\}$ lie in the interior of $\text{conv}(Q)$. If $\delta > 0$ is sufficiently small, then the farthest points from a_i in Q are b_i, c_i, and d_i, for every $i = 1, 2, \ldots, k$.

Every non-crossing tour of Q visits the convex hull vertices in the cyclic order determined by $\text{conv}(Q)$. We obtain a non-crossing tour by replacing some edges of $\text{conv}(Q)$ with non-crossing paths visiting the points lying in the interior of $\text{conv}(Q)$. If we replace either edge $b_i c_i$ or $c_i d_i$ with the path (b_i, a_i, c_i) or (c_i, a_i, d_i), respectively, for every $i = 1, 2, \ldots, k$, then we obtain a tour. Let $\mathcal{H}$ be the set of 2^k tours obtained in this way. These tours are non-crossing, since for every i, we exchange an edge of Δ_i with a path lying in Δ_i, and the deltoids Δ_i are interior disjoint. The tours in $\mathcal{H}$ have the same length, $L = |\text{conv}(Q)| - k\delta + 2\sum_{i=1}^{k} |a_i c_i|$, since $|a_i b_i| = |a_i d_i| = |a_i c_i|$. In the full paper [10], we show that this length is maximal over all non-crossing tours (cycles).

To obtain the asserted bound, we use a skinny hexagon (instead of deltoid Δ_i) with five equidistant vertices on a circle centered at a_i. We now have four possible ways to insert each a_i into the tour, which implies $\texttt{sc}_{\max}(n) = \Omega(4^{n/6}) = \Omega(2^{n/3})$. ◀

Typically for the longest matching, spanning tree or spanning cycle, one expects to see many crossings. Somewhat surprisingly, we show that this is not always the case.

▶ **Corollary 6.** *For every even $n \geq 2$, there exists an n-element point set (in general position) whose longest perfect matching is non-crossing.*

Longest spanning trees and shortest spanning cycles. We state without proof our results on the maximum multiplicity $\texttt{st}_{\max}(n)$ of the longest crossing-free spanning tree on points, and the maximum multiplicity $\texttt{sc}_{\min}(n)$ of the shortest non-crossing Hamiltonian cycle on n points.

▶ **Theorem 7.** *The vertex set of a regular convex n-gon admits $\Omega(2^n)$ longest non-crossing spanning trees. Consequently, $\texttt{st}_{\max}(n) = \Omega(2^n)$.*

▶ **Theorem 8.** *Let $\texttt{sc}_{\min}(n)$denote the maximum number of shortest tours that an n-element point set can have.*

(i) *If S is a set of $n \geq 3$ points in convex position, then $\texttt{sc}_{\min}(S) = 1$.*

(ii) *For points in general position, we have $\texttt{sc}_{\min}(n) \geq 2^{\lfloor n/3 \rfloor}$.*

Recall that a geometric graph $G = (V, E)$ is called a *(geometric) thrackle*, if any two edges in E either cross or share a common endpoint.

▶ **Theorem 9.** *Let* $\mathtt{tc}_{\max}(n)$ *denote the maximum number of longest tours that an* n*-element point set in convex position can have. For* n *odd we have* $\mathtt{tc}_{\max}(n) = 1$ *and the (unique) longest tour is a thrackle. For* n *even we have* $\mathtt{tc}_{\max}(n) = n/2$.

References

1 O. Aichholzer, T. Hackl, B. Vogtenhuber, C. Huemer, F. Hurtado and H. Krasser, On the number of plane geometric graphs, *Graphs and Combinatorics* **23(1)** (2007), 67–84.

2 M. Ajtai, V. Chvátal, M. Newborn and E. Szemerédi, Crossing-free subgraphs, *Annals Discrete Math.* **12** (1982), 9–12.

3 P. Beame, M. Saks and J. Thathachar, Time-space tradeoffs for branching programs, *Proc. 39th Annual Symposium on Foundations of Computer Science*, 1998, pp. 254–263.

4 K. Buchin, C. Knauer, K. Kriegel, A. Schulz, and R. Seidel, On the number of cycles in planar graphs, *Proc. 13th COCOON*, vol. 4598 of LNCS, Springer, 2007, pp. 97–107.

5 K. Buchin and A. Schulz, On the number of spanning trees a planar graph can have, *Proc. 18th European Symposium on Algorithms*, vol. 6346 of LNCS, Springer, 2010, pp. 110-121.

6 E. Demaine, Simple polygonizations, `http://erikdemaine.org/polygonization/`.

7 A. Dumitrescu, On two lower bound constructions, *Proc. 11th Canadian Conf. on Computational Geometry*, 1999, pp. 111-114.

8 A. Dumitrescu, On the maximum multiplicity of some extreme geometric configurations in the plane, *Geombinatorics* **12(1)** (2002), 5–14.

9 A. Dumitrescu and R. Kaye, Matching colored points in the plane: some new results, *Computational Geometry: Theory and Applications* **19(1)** (2001), 69–85.

10 A. Dumitrescu, A. Schulz, A. Sheffer, and Cs. D. Tóth, Bounds on the maximum multiplicity of some common geometric graphs, `arxiv.org/abs/1012.5664`.

11 A. Dumitrescu and W. Steiger, On a matching problem in the plane, *Discrete Mathematics* **211(1-3)** (2000), 183–195.

12 P. Flajolet and M. Noy, Analytic combinatorics of non-crossing configurations, *Discrete Mathematics* **204** (1999), 203–229.

13 A. García, M. Noy and A. Tejel, Lower bounds on the number of crossing-free subgraphs of K_N, *Computational Geometry: Theory and Applications* **16(4)** (2000), 211–221.

14 M. Hoffmann, M. Sharir, A. Sheffer, C. Tóth, and E. Welzl, Counting plane graphs: flippability and its applications, 2010, manuscript. `arXiv:1012.0591`

15 F. Hurtado and M. Noy, Counting triangulations of almost-convex polygons, *Ars Combinatoria* **45** (1997), 169–179.

16 M. Newborn and W. O. J. Moser, Optimal crossing-free Hamiltonian circuit drawings of K_n, *J. Combin. Theory Ser. B* **29** (1980), 13–26.

17 A. Razen, J. Snoeyink, and E. Welzl, Number of crossing-free geometric graphs vs. triangulations, *Electronic Notes in Discrete Mathematics* **31** (2008), 195–200.

18 F. Santos and R. Seidel, A better upper bound on the number of triangulations of a planar point set, *Journal of Combinatorial Theory Ser. A* **102(1)** (2003), 186–193.

19 M. Sharir and E. Welzl, On the number of crossing-free matchings, cycles, and partitions, *SIAM J. Comput.* **36(3)** (2006), 695–720.

20 M. Sharir and E. Welzl, Random triangulations of point sets, *Proc. 22nd Annual ACM-SIAM Symposium on Computational Geometry*, ACM Press, 2006, pp. 273–281.

21 M. Sharir and A. Sheffer, Counting triangulations of planar point sets, manuscript 2010, `arxiv.org/abs/0911.3352`.

On the computational complexity of Ham-Sandwich cuts, Helly sets, and related problems

Christian Knauer*[1], Hans Raj Tiwary[2], and Daniel Werner†[3]

1 **Institut für Informatik, Universität Bayreuth, Germany,**
`christian.knauer@uni-bayreuth.de`

2 **Département de Mathématique, Université Libre de Bruxelles, Belgium,**
`hans.raj.tiwary@ulb.ac.be`

3 **Institut für Informatik, Freie Universität Berlin, Germany,**
`daniel.werner@fu-berlin.de`

Abstract

We study several canonical decision problems arising from some well-known theorems from combinatorial geometry. Among others, we show that computing the minimum size of a *Caratheodory set* and a *Helly set* and certain decision versions of the *ham-sandwich cut problem* are W[1]-hard (and NP-hard) if the dimension is part of the input. This is done by fpt-reductions (which are actually ptime-reductions) from the d-Sum problem. Our reductions also imply that the problems we consider cannot be solved in time $n^{o(d)}$ (where n is the size of the input), unless the Exponential-Time Hypothesis (ETH) is false.

The technique of embedding d-Sum into a geometric setting is conceptually much simpler than direct fpt-reductions from purely combinatorial W[1]-hard problems (like the clique problem) and has great potential to show (parameterized) hardness and (conditional) lower bounds for many other problems.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases computational geometry, combinatorial geometry, ham-sandwich cuts, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.649

1 Introduction

Many theorems from combinatorial geometry are of the following type: If a set of n objects has a certain property, then there is already a subset of size $d+1$ that has this property. Two examples of this are Caratheodory's Theorem [6] and Helly's Theorem [22].

Caratheodory's Theorem states, in one of its several formulations, that whenever a point p is contained in the convex hull of a point set in $\mathbb{R}^d$, then it is already contained in the convex hull of a subset of size at most $d+1$. A minimal set containing p in the convex hull is called a *Caratheodory set* for p. The canonical decision problem, that asks whether there is an even smaller set, can be stated as follows:

* This research was supported by the German Science Foundation (DFG) under grant Kn 591/3-1.

† This research was funded by Deutsche Forschungsgemeinschaft within the Research Training Group (Graduiertenkolleg) "Methods for Discrete Structures".

▶ **Definition 1.** (d-Caratheodory-Set) Given a point set in $\mathbb{R}^d$, are there d points whose convex hull contains the origin?

Stated in a dual setting, this gives another well known theorem: If n convex sets in $\mathbb{R}^d$ have an empty intersection, then by Helly's Theorem there are already $d+1$ whose intersection is empty. This leads to the following decision problem:

▶ **Definition 2.** (d-Helly-Set) Given n convex sets $P_1, \ldots, P_n$ in $\mathbb{R}^d$, do any d of them have an empty intersection?

The canonical decision versions of Caratheodory's and Helly's Theorem have not explicitly been considered in the literature so far. This is quite surprising, as they are interesting to people from computational as well as discrete geometry. However, similar problems arise in the context of Linear Programming, most notably the following:

▶ **Definition 3.** (d-Min-IIS) Given n inequalities in $\mathbb{R}^d$, do any d of them have an empty intersection?

The d-Min-IIS has been studied before, mainly because of its connection to the NP-complete Maximum-Feasible-Subsystem problem, where one is given an infeasible linear program and one has to find a feasible subsets of constraints of maximum size. Amaldi et al. [2] show that d-Min-IIS is NP-hard by a (transitive) reduction from Dominating-Set. However, the dimension depends on the size of the graph, so it does not reveal anything with respect to this parameter d.

The Ham-Sandwich Theorem as a corollary of the Borsuk-Ulam Theorem (see, e.g., Matoušek [29]) states that for any d finite point sets in $\mathbb{R}^d$ there is a hyperplane that bisects all of the sets at once, i.e., has at most half of the points on each side. Computing a ham-sandwich cut efficiently is an important problem and has been studied extensively (see Edelsbrunner and Waupotitsch [13], Matoušek et al. [27], Yu [36]). For general dimension, the fastest known algorithm [27] runs in time roughly $O\left(n^{d-1}\right)$.

The ham-sandwich problem is not a decision problem, as, given an instance, we know that there always exists a solution, but still it is not known how to find it efficiently. Such problems are captured by the complexity class PPAD, see Papadimitrou [33]. It is an important open question whether computing a ham-sandwich cut is PPAD complete. In this paper we show that a natural "incremental" approach for computing the ham-sandwich cut will not work unless $W[1] = P$: One way to find a ham-sandwich cut incrementally could be to take any point, decide whether there is some ham-sandwich cut through it, and perform a dimension reduction until the hyperplane is determined. This gives rise to the following decision problem:

▶ **Definition 4.** (d-Ham-Sandwich) Given d sets $P_1, \ldots, P_d$ in $\mathbb{R}^d$ and a point $a \in \mathbb{R}^d$, is there a ham-sandwich cut that passes through a?

We show that d-Ham-Sandwich is $W[1]$-hard and therefore most likely no polynomial algorithm (FPT or otherwise) exists for this problem.

The reductions presented in this paper use a new technique of embedding of d-Sum into the d-dimensional space. Thereto, a d-Sum instance is encoded into sets of points (or hyperplanes, respectively), and the property of d elements summing up to 0 is expressed by an equivalent geometric property of the point set, e.g., allowing a ham-sandwich cut through the origin.

1.1 Overview

The main results of this paper presented in In Sec. 3, 4, and 5 are the following:

▶ **Theorem 5.** *The problems d-*CARATHEODORY-SET *and d-*HELLY-SET *are W[1]-hard with respect to the parameter d and NP-hard.*

All reductions are slight modifications of the hardness proof for the problem d-AFFINE-CONTAINMENT considered in Sec. 2.

Subsequently, two easy corollaries are derived from these theorems:

▶ **Corollary 6.** *The problem d-*MIN-IIS *is W[1]-hard with respect to the dimension.*

Observe that this problem becomes polynomial-time solvable if we ask for $d+1$ halfspaces by first solving the corresponding linear program and afterwards applying Helly's Theorem.

▶ **Corollary 7.** *Deciding whether a point q is in general position*[1] *with respect to P is W[1]-hard with respect to d and NP-hard.*

For the d-HAM-SANDWICH problem, a little more work has to be done. By adding certain *balancing points* to the previous construction, it is achieved that ham-sandwich cuts through the origin correspond exactly to sets of d numbers that sum up to 0. From this construction, the next result follows:

▶ **Theorem 8.** *The d-*HAM-SANDWICH *problem is W[1]-hard with respect to the dimension and NP-hard.*

Combining our reductions with a result of Pătraşcu and Williams [34], Theorems 5 and 8 immediately give:

▶ **Corollary 9.** *The problems d-*CARATHEODORY-SET*, d-*HELLY-SET *and d-*HAM-SANDWICH *cannot be solved in time $n^{o(d)}$ (where n is the size of the input), unless the Exponential-Time Hypothesis (ETH) is false*[2].

1.2 Related work

The study of computational variants of theorems from discrete geometry is not new. Several problems that arise from theorems in discrete geometry have received a lot of attention, most notably computation of (approximate) center- and Tverberg points in the plane as well as in higher dimension. In the plane, surprisingly one can compute a centerpoint in linear time [24]. In three dimensions, $O(n^2\text{polylog } n)$ deterministic algorithms are known ([31], [10]). If the dimension is part of the input, the best (randomized) algorithm due to Chan [7] runs in $O(n^{d-1})$ time. The corresponding decision problem has also been considered, i.e., to decide whether a given point is a center point. This problem has been shown to be co-NP complete if d is part of the input by Teng [35]. See also Agarwal et al. [1] and Miller and Sheehy [30] for recent progress.

A decision version of ham-sandwich problem in the plane has been studied by Chien and Steiger [9]: decide whether there is more than one cut. They provide an $\Omega(n \log n)$ lower

[1] No hyperplane that contains d points from P also contains q.

[2] The Exponential Time Hypothesis [23] conjectures that n-variable 3-CNFSAT cannot be solved in $2^{o(n)}$-time.

bound, which shows that searching for an object can be easier than deciding whether an object is unique.

Perhaps surprisingly, the computation of smallest sets arising from Caratheodory's and Helly's Theorem has not been explicitly studied even though it has been studied under the guise of IIS in the context of Linear Programming.

Even though the dimension of geometric problems is a natural parameter for studying their parameterized complexity, only relatively few results of this type are known: Langerman and Morin [26] gave fpt-algorithms for the problem of covering points with hyperplanes, while the problem of computing the volume of the union of axis parallel boxes has been shown to be W[1]-hard by Chan [8]. Cabello et al. [5, 4] have developed a technique that has been applied succesfully to show W[1]-hardness for a number of problems from various application areas like shape matching [3], clustering [4, 19], and discrepancy-computation [20]. We refer to Giannopoulos et al. [21] and Knauer [25] for surveys on other parameterized complexity results for geometric problems.

For a general introduction to combinatorial geometry, we recommend Matoušek [28] and Ziegler [37].

1.3 Parameterized complexity

Parameterized complexity theory provides a framework for the study of algorithmic problems by measuring their complexity in terms of one or more parameters, explicitly or implicitly given by their underlying structure, in addition to the problem input size. For an introduction to the field of parameterized complexity theory, we refer to the textbooks of Flum and Grohe [17], Niedermeier [32] and Downey and Fellows [12].

The dimension d of geometric problems in $\mathbb{R}^d$ is a natural parameter for studying their parameterized complexity. In terms of parameterized complexity theory the question is whether these problems are *fixed-parameter tractable* with respect to d. Proving a problem to be W[1]-hard with respect to d, gives a strong evidence that an fpt-algorithm (i.e., an algorithm that runs in time $O\left(f(d) \cdot n^c\right)$ for some fixed c and an arbitrary function f) does not exist. W[1]-hardness is often established by fpt-reductions from the clique problem in general graphs, which is known to be W[1]-complete [12]. Below we use a different approach by giving *conceptually much simpler* fpt-reductions from the d-Sum problem [18, 15]:

▶ **Definition 10.** (d-Sum) Given n integers, are there d (not necessarily distinct) numbers that sum up to 0?

This problem is NP-hard [15] and can be solved in (roughly) $O(n^{d/2})$ time. It can be shown to be W[1]-hard with respect to d from a simple reduction from the subset-sum problem which was shown to be W[1]-hard by Downey and Koblitz [16]. Recently it has been shown [34] (without using parameterized complexity explicitly) that, unless the ETH fails, d-Sum cannot be solved in time $n^{o(d)}$.

Reductions from 3-Sum seem somewhat more "natural" for computational geometers: Gajentaan and Overmars [18] introduced the 3-Sum problem for the purpose of arguing that certain problems in planar geometry "should" take $\Omega(n^2)$ time; showing 3-Sum-hardness for such problems is considered a routine task today. Knauer [25] has pointed out that the work of Erickson [15] implicitly shows W[1]-hardness for two geometric problems parameterized by the dimension (the affine degeneracy-detection problem and the convex hull simlicity-detection problem) by giving reductions from the k-Sum problem. Surprisingly – apart from Erickson's work – this technique has not been used to show W[1]-hardness of more geometric problems in $\mathbb{R}^d$.

1.4 Basic notation

For a hyperplane h and a point set P, let h_P^+ denote the set of points of P that lie strictly on the positive side of h, and analogously h_P^-. For a point p, by $(p)_i$ we denote the i–th coordinate of p. Finally, for a number x as usual let

$$\operatorname{sign}(x) := \begin{cases} 1 & x \geq 0 \\ -1 & x < 0. \end{cases}$$

2 Affine Containment

We start with a problem for which we think the hardness proof is the most straightforward. This proof will subsequently be modified to show the main theorems.

▶ **Definition 11.** (d-Affine-Containment) Given a set of points P in $\mathbb{R}^d$, is the origin contained in the affine hull of any d points?

Recall that $x \in \operatorname{affHull}(\{p_1, \ldots, p_j\})$ iff there exist $\alpha_i, 1 \leq i \leq j$ such that $\sum \alpha_i = 1$ and $\sum \alpha_i p_i = x$.

For a given set $S = \{s_1, \ldots, s_n\}$, we will create a point-set in $\mathbb{R}^{d+1}$ in which $d+1$ points span an affine plane through the origin if and only d of these numbers sum up to 0.

Let e_i denote the i–th unit vector. Set

$$p_i^j := \frac{1}{s_i} \cdot e_j + e_{d+1} = \left(0, \ldots, \frac{1}{s_i}, \ldots, 0, \ldots, 1\right)^T$$

and $q := -\sum_{i=1}^d e_i$.

The set P consists of all points $p_i^j, 1 \leq j \leq d, 1 \leq i \leq n$ and the point q. The size of the point set is thus $n \cdot d + 1$.

▶ **Lemma 12.** *There are d elements that sum up to 0 iff there are $d+1$ points in P whose affine hull contains the origin*[3].

Proof. $\Rightarrow$: Let $\sum_{j=1}^d s_{i_j} = 0$. We choose points $x_j = p_{i_j}^j$, $1 \leq j \leq d$ and $x_{d+1} = q$. Let $\alpha_j = s_{i_j}$ and $\alpha_{d+1} = 1$. Then

$$\sum_{j=1}^{d+1} \alpha_j x_j = \sum_{j=1}^{d} s_{i_j} p_{i_j}^j + q = \sum_{j=1}^{d} e_j + \left(\sum_{j=1}^{d} s_{i_j}\right) e_{d+1} - \sum_{j=1}^{d} e_j = \mathbf{0}$$

and

$$\sum_{j=1}^{d+1} \alpha_j = \sum_{j=1}^{d} s_{i_j} + \alpha_{d+1} = 1.$$

That means that $\mathbf{0}$ is in $\operatorname{affHull}\left(\{p_{i_1}^1, \ldots p_{i_d}^d, q\}\right)$.

$\Leftarrow$: Let $\mathbf{0} \in \operatorname{affHull}(\{x_1, \ldots, x_d\})$, i.e., let $\sum_{j=1}^{d+1} \alpha_j x_j = \mathbf{0}$ and $\sum \alpha_j = 1$. As all points but q lie on the hyperplane $x_{d+1} = 1$, one of the points, without loss of generality x_{d+1}, is q.

[3] Recall that the dimension is also $d+1$.

Because of $(q)_{d+1} = 0$, and $(x)_{d+1} = 1$ for all $x \neq q$, by computing the $(d+1)$-st coordinate we get

$$0 = \sum_{j=1}^{d} (\alpha_j x_j)_{d+1} = \sum_{j=1}^{d} \alpha_j (x_j)_{d+1} = \sum_{j=1}^{d} \alpha_j \tag{1}$$

and thus $\alpha_{d+1} = 1 - \sum_{j=1}^{d} \alpha_j = 1$.

Further, as $\sum_{j=1}^{d+1} \alpha_j x_j = \mathbf{0}$, the other points satisfy

$$\sum_{j=1}^{d} \alpha_j x_j = -\alpha_{d+1} q = \sum_{j=1}^{d} e_j.$$

Any x_j is nonzero for only one other coordinate except the $(d+1)$-st, and as $(q)_j = -1$ for all $j < d+1$, for each j there is at least one point that is nonzero at coordinate j (in particular, also $\alpha_j \neq 0$). Thus, there are exactly d such points. Without loss of generality assume that x_j is the point that is nonzero in coordinate j, so $(x_j)_j = \frac{1}{s_{i_j}}$ for some i_j. This means that $\alpha_j \frac{1}{s_{i_j}} - 1 = 0$, and thus $\alpha_j = s_{i_j} \in S$, which implies (Eqn. 1) that we have d elements in S summing up to 0. ◀

▶ **Theorem 13.** *d-AFFINE-CONTAINMENT is W[1]-hard with respect to the dimension and NP-hard.*

3 Caratheodory sets

In order to use the previous construction to prove the first part of Theorem 5, we have to modify it such that all coefficients can be chosen positive. Observe that $\mathbf{0} \in \text{conv}(P)$ iff $\mathbf{0} = \sum_{p \in P} \alpha_p p$ for any $\alpha_p \geq 0, \sum \alpha_p > 0$ (proof: divide by $\sum \alpha_p$). To this end we now define

$$p_i^j = \frac{1}{|s_i|} \cdot e_j + \text{sign}(s_i) \cdot e_{d+1}$$

and q as above. The set P again consists of all the points $p_i^j, 1 \leq j \leq d, 1 \leq i \leq n$ and q.

▶ **Lemma 14.** *There are d elements in S that sum up to 0 iff the origin lies in the convex hull of $d+1$ points of P.*

Proof. $\Rightarrow$: Let $\sum_{j=1}^{d} s_{i_j} = 0$. Setting $\alpha_j = |s_{i_j}| > 0, x_j = p_{i_j}^j$ for $1 \leq j \leq d$ and $\alpha_{d+1} = 1$, $x_{d+1} = q$ again yields

$$\sum_{j=1}^{d+1} \alpha_j x_j = \sum_{j=1}^{d} |s_{i_j}| p_{i_j}^j + q = \sum_{j=1}^{d} e_j + \left(\sum_{j=1}^{d} \text{sign}(s_{i_j}) |s_{i_j}| \right) e_{d+1} - \sum_{j=1}^{d} e_j = \mathbf{0}.$$

$\Leftarrow$: Let $\sum_{j=1}^{d+1} \alpha_j x_j = \mathbf{0}, \alpha_j \geq 0$. As all points lie in the positive halfspace $\sum^d e_j^* x > 0$, q is one of the points of the convex combination. We can assume $x_{d+1} = q$ and $\alpha_{d+1} = 1$. Further, by the same argument as in Lemma 12, there are at least d other points for the total sum to become $\mathbf{0}$. Again, without loss of generality let $(x_j)_j \neq 0$. As $(q)_j = -1$ for all $1 \leq j \leq d$, this means that $\alpha_j \frac{1}{|s_{i_j}|} = 1$ for some i_j, and thus $\alpha_j = |s_{i_j}|$. Further, because of the $(d+1)$-st coordinate, we get

$$0 = \sum_{j=1}^{d} \alpha_j \text{sign}(s_{i_j}) = \sum_{j=1}^{d} \text{sign}(s_{i_j}) \cdot |s_{i_j}| = \sum_{j=1}^{d} s_{i_j}$$

and thus we have d elements summing up to 0. ◀

Thereby we have shown the first part of Theorem 5.

3.1 Remark

Observe that if we project all points onto the unit sphere, all the above properties still hold: Clearly, $\mathbf{0} \in \operatorname{conv}(P)$ iff $\mathbf{0} \in \operatorname{conv}\left(\pi_{S^{d-1}}(P)\right)$. Thus, we can even assume all points to lie in convex position and thereby get a slightly stronger result:

▶ **Theorem 15.** *The following problem is W[1]-hard and NP-hard: Given a V-polytope in $\mathbb{R}^d$, is the origin contained in the convex hull of any d vertices?*

4 Helly sets

Starting from the result in the previous section, we will now show how to prove the hardness for the d-Helly-Set problem. Using a duality transform, for a given set P in $\mathbb{R}^d$, we will construct a set of convex sets (that are actually half-spaces) such that d have an empty intersection if and only if there are d points in P that contain the origin in their convex hull. A similar construction (which is used to prove Caratheodory's Theorem from Helly's Theorem) can be found in [14, Chapter 2.3].

Consider a set P of points $p_1, \cdots, p_n \in \mathbb{R}^d$ whose convex hull contains the origin. For each point $p \in P$ set consider the halfspace

$$p^* = \left\{x \mid p^T x \geq 1\right\}.$$

Define P^* to be the set of all these halfspaces corresponding to the points in P. We show that any Caratheodory set of P (for the origin) corresponds to a Helly set (a set of halfspaces with empty intersection) of P^* of the same size. Since checking if the minimum Caratheodory set has cardinality at most d is W[1]-hard, it then follows that checking if the minimum Helly set is of cardinality at most d is also $W[1]$-hard.

Let $Q \subseteq P$ and let V be a $d \times |Q|$ matrix whose columns represent the vectors in Q. Further, let $\operatorname{cone}(V)$ denote the conic hull of the vectors, i.e., the set $\left\{\sum_{q\in Q} \alpha_q q \mid \alpha_q \geq 0\right\}$.

Using the fact that $\operatorname{cone}(V)$ is pointed if and only if $V^T x \leq \mathbf{0}$ is a full-dimensional cone, we can now show the main lemma of this section, which is a variant of Gordan's Theorem, see e.g. Dantzig and Thapa [11, Theorem. 2.13]:

▶ **Lemma 16.** *Let $Q \subseteq P$ and let V be a $d \times |Q|$ matrix whose columns represent the vectors in Q. Then $\mathbf{0} \in conv(V)$ if and only if the system of inequalities $V^T x \geq \mathbf{1}$ is infeasible.*

Proof. $\Rightarrow$: Suppose that $V^T x \geq \mathbf{1}$ is feasible. Then there exists a vector $\alpha \in \mathbb{R}^d$ such that $V^T \alpha \leq -\mathbf{1}$. That is, $V^T \alpha < \mathbf{0}$ and thus $V^T x \leq \mathbf{0}$ is a full-dimensional cone. Therefore, $\operatorname{cone}(V)$ is pointed. But this means that $\mathbf{0} \notin \operatorname{conv}(V)$.

$\Leftarrow$: Now suppose $\mathbf{0} \notin \operatorname{conv}(V)$, then $\operatorname{cone}(V)$ is pointed and therefore $V^T x \leq \mathbf{0}$ is a full-dimensional cone. Thus, there exists $\alpha \in \mathbb{R}^d$ such that $V^T \alpha < \mathbf{0}$, and so for a large enough $\lambda > 0$, $V^T(-\lambda\alpha) > \mathbf{1}$ and hence $V^T x \geq \mathbf{1}$ is feasible. ◀

Thus, any set $Q \subseteq P$ of points whose convex hull contains the origin corresponds to a set $Q^* \subseteq P^*$ of convex set (inequalities) of the same size that has an empty intersection, and vice versa. This finishes the proof of the second part of Theorem 5.

As the convex sets in this case are even halfspaces, we can derive the stronger result of Corollary 6.

5 Ham-Sandwich cuts

Using the construction from Sec. 2, we will now prove that the decision version of the ham-sandwich problem is W[1]-hard.

A hyperplane h is said to *bisect* a set Q if $|h_Q^+| \leq \left\lfloor \frac{|Q|}{2} \right\rfloor$ and $|h_Q^-| \leq \left\lfloor \frac{|Q|}{2} \right\rfloor$. A *ham-sandwich cut* of d point sets $P_1, \ldots, P_d$ in $\mathbb{R}^d$ is a hyperplane h that bisects each of the sets. In particular, if the number of points in each set is odd, the hyperplane has to pass through at least one of the points from each set.

Def. 4 asks whether there is a cut that goes through a given point a. Via translation we can obviously assume a to be the origin. This will be called a *linear ham-sandwich cut*.

In order to show Theorem 8 we will create $d+1$ sets $P_1, \ldots, P_{d+1}$. The set P_{d+1} will consist of the single point $q = \sum_{j=1}^{d} e_j$ (which is $-q$ in the above notion). The sets P_j will be the union of the two set R_j and B_j. R_j contains all points of the form p_i^j, defined exactly as in Sec. 2, i.e.,

$$R_j := \left\{ p_i^j \mid 1 \leq i \leq n \right\}.$$

for $p_i^j = \frac{1}{s_i} e_j + e_{d+1}$. If we choose a linear hyperplane through one of these points, the number of points on each side will (most likely) not be the same. So in addition to these, for each of these sets we need $n-1$ *balancing* points B_j to ensure that any linear hyperplane passing through any of these points has equally many points of P_j on both sides (c.f. Figure 1). Thus, the set $P = \bigcup P_j$ is of size $d\,(2n-1)+1$.

5.1 Construction of the Balancing-set

The idea is to add a point set similar to the mirror image of the original set R_j. This way any hyperplane that has *many* of the original points on, say, the positive side, will contain *few* of the mirrored points on the positive side, and vice versa.

By making the total number of points in each set P_j odd, we will ensure that any ham-sandwich cut must pass through one of the points from P_j. Further, by the construction of the balancing set, it will not be possible to choose a linear cut through q that also goes through any of these balancing points, thereby getting the correspondence between subsets of S and linear cuts through q.

For this, we will choose the mirror-image of a set of $n-1$ points that lie *between two successive points* in R_j (recall that all points from R_j lie on a line; this is why we use the construction from Sec. 2). Thereto, let S be in ascending order with respect to $s_i \prec s_j$ iff $1/s_i < 1/s_j$ (or, equivalently: $1/s_i < 1/s_j$ for $i < j$).

Then, let $\varepsilon_j = \frac{1}{2^j}$ and

$$b_i^j := -\left(\frac{1}{s_i - \varepsilon_j}\right) \cdot e_j - e_{d+1}.$$

This the mirror image of a point slightly to the right of p_i^j, for $1 \leq i < n$; see Figure 1. Let B_j consist of all balancing points of the form b_i^j and set

$$P_j := R_j \cup B_j.$$

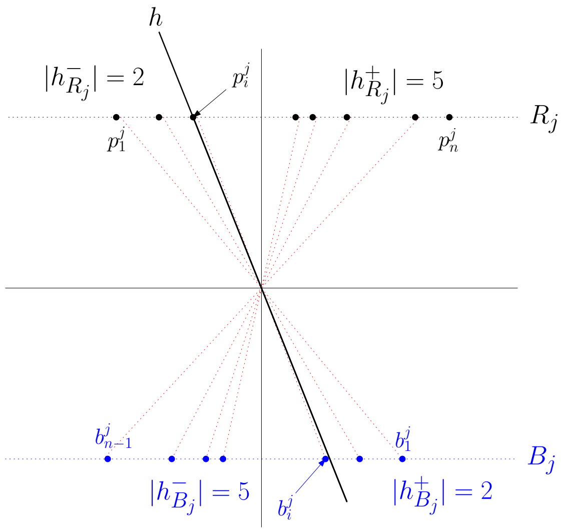


Figure 1 The set P_j: points and balancing points

5.2 The main lemma

Now we come to prove the main lemma, namely that the point set allows a linear ham-sandwich cut if and only if there are d elements that sum up to 0, based on the following two simple lemmas. The first one states that any (not necessarily linear) ham-sandwich cut intersects exactly one point from each set P_j, whereas the second one guarantees that any linear hyperplane that contains a point from R_j will bisect P_j.

▶ **Lemma 17.** *Any linear ham-sandwich cut intersects exactly one point from each P_j, $1 \leq j \leq d+1$.*

Proof. For $P_{d+1} = \{q\}$ this is trivial. We show that for any linear ham-sandwich cut $h = (h_1, \ldots, h_{d+1})$ we have $h_i \neq 0$ for all i: First, if h_{d+1} were 0, because the cut must pass through at least one point from each set, we would have $h_j = 0$ for all j. Thus, $h_{d+1} \neq 0$. Further, as $h_j(p^j)_j = -h_{d+1}(p^j)_j \neq 0$ for some $p^j \in P_j$, also $h_j \neq 0$ for all j.

Thus, no cut can pass through more than one point of any set P_j: If

$$h_j(p)_j + h_{d+1}(p)_{d+1} = h \cdot p = 0 = h \cdot p' = h_j(p')_j + h_{d+1}(p')_{d+1}$$

for two points $p, p' \in P_j$, then $p = p'$ or $h_j = 0$, a contradiction. ◀

▶ **Lemma 18.** *Any linear hyperplane intersecting a single point from R_j bisects the set P_j.*

Proof. Let $h \cdot p^j_i = 0$ and without loss of generality $h \cdot p^j_k < 0$ for all $1 \leq k < i$. Then also $h \cdot -b^j_k < 0$ and thus $h \cdot b^j_k > 0$ for all $1 \leq k < i$. Further, $h \cdot p^j_k > 0$ for all $k > i$ and $h \cdot b^j_k < 0$ for $k \geq i$. So

$$|h^-_{P_j}| = |h^-_{R_j}| + |h^-_{B_j}| = i - 1 + n - i = \left\lfloor \frac{|P_j|}{2} \right\rfloor = |h^+_{P_j}|.$$

◀

▶ **Lemma 19.** *There are d elements in S that sum up to 0 if and only if there is a linear ham-sandwich cut.*

Proof. $\Rightarrow$: Let $\sum_{j=1}^{d} s_{i_j} = 0$. We have to find a linear hyperplane $h \cdot x = 0$ such that for each set P_j it holds that $|h^+_{P_j}|, |h^-_{P_j}| \leq \left\lfloor \frac{|P_j|}{2} \right\rfloor$. Choose $h_j = s_{i_j}$ for $1 \leq j \leq d$ and $h_{d+1} = -1$. Because $\sum^d s_{i_j} = 0$, we have $h \cdot q = \sum^d s_{i_j} = 0$ (so the one element set P_{d+1} is bisected). Further,

$$h \cdot p^j_{i_j} = h_j \cdot 1/s_{i_j} + h_{d+1} \cdot 1 = 1 - 1 = 0.$$

Because of Lemma 18, this means that all sets are bisected, and thus we have a linear ham-sandwich cut.

$\Leftarrow$: Let h be a linear ham-sandwich cut. All h_i are nonzero (Lemma 17), so we can assume $h_{d+1} = -1$. For each j, we have $h \cdot p^j = 0$ for exactly one point $p^j \in P_j$. This means that

$$0 = h \cdot p^j = h_j(p^j)_j + h_{d+1}(p^j)_{d+1} = h_j(p^j)_j - 1(p^j)_{d+1} = h_j(p^j)_j - 1,$$

and so either $h_j = s_{i_j}$ or $h_j = s_{i_j} - \varepsilon_j$ for some i_j. Because for any $\emptyset \neq J \subset \{1, \ldots, d\}$ we have $0 < \sum_{j \in J} \varepsilon_j < 1$ and S is a set of integers, if one (or more) of the h_j were of the latter form, the total sum can never be an integer, and in particular not 0. But this is required for q to lie on h.

Thus, $h_j = s_{i_j} \in S$ for some i_j, and as q also lies on the hyperplane, we get

$$0 = hq = \sum_{j=1}^{d} h_j = \sum_{j=1}^{d} s_{i_j},$$

i.e., there are d elements in S that sum up to 0. ◀

From this Theorem 8 follows.

5.3 Remarks

In the previous construction, the origin (i.e., the point for which we want to solve the decision version) is not part of any of the sets. This is easily fixed: Set $P_{d+1} = \{\mathbf{0}, q/2, q\}$. Then any ham-sandwich cut through $\mathbf{0}$ also has to go through the other two points (otherwise there would be too many points on the one side). Thus it also contains q. On the other hand, whenever there are no such d elements that sum up to 0, all ham-sandwich cuts are (truly) affine hyperplanes through $q/2$. This gives a slightly stronger result:

▶ **Corollary 20.** *The following problem is W[1]-hard with respect to the dimension and NP-hard: Given d point sets in $\mathbb{R}^d$ and a point $a \in \bigcup P_i$, is there a ham-sandwich cut through a?*

For a given family of $d + 1$ sets in $\mathbb{R}^d$ we are not guaranteed that there is a cut that bisects all the sets simultaneously. By adding the origin as a single set, the previous shows that deciding whether there is still such a cut is also a computationally hard question:

▶ **Corollary 21.** *The following problems are W[1]-hard with respect to the dimension and NP-hard:*

(d-Strong-Ham-Sandwich) Given $d + 1$ point sets in $\mathbb{R}^d$, is there a hyperplane that bisects all sets?

Acknowledgements. We thank the anonymous reviewers for their helpful comments.

References

1 P. K. Agarwal, M. Sharir, and E. Welzl. Algorithms for center and Tverberg points. In *Journal of the ACM*, volume 5, pages 1–20, New York, NY, USA, 2008. ACM.

2 E. Amaldi, M. E. Pfetsch, and L. E. T. Jr. On the maximum feasible subsystem problem, iiss and iis-hypergraphs. *Math. Program.*, 95(3):533–554, 2003.

3 S. Cabello, P. Giannopoulos, and C. Knauer. On the parameterized complexity of d-dimensional point set pattern matching. In *Proc. 2nd Int. Workshop on Parameterized and Exact Computation (IWPEC)*, volume 4169 of *LNCS*, pages 175–183, 2006.

4 S. Cabello, P. Giannopoulos, C. Knauer, D. Marx, and G. Rote. Geometric clustering: fixed-parameter tractability and lower bounds with respect to the dimension. *ACM Transactions on Algorithms*, 2009. to appear.

5 S. Cabello, P. Giannopoulos, C. Knauer, and G. Rote. Geometric clustering: fixed-parameter tractability and lower bounds with respect to the dimension. In *Proc. 19th Ann. ACM-SIAM Sympos. Discrete Algorithms*, pages 836–843, 2008.

6 C. Caratheodory. Über den Variabilitätsbereich der Fourierschen Konstanten von positiven harmonischen Funktionen. *Rend. Circ. Mat. Palermo*, 32:193–217, 1911.

7 T. M. Chan. An optimal randomized algorithm for maximum tukey depth. In *Proceedings of the fifteenth annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '04, pages 430–436, 2004.

8 T. M. Chan. A (slightly) faster algorithm for Klee's measure problem. In *Proc. 24th Annual Symposium on Computational Geometry*, pages 94–100, 2008.

9 H. Chien and W. Steiger. Some geometric lower bounds. In *In ISAAC: 6th International Symposium on Algorithms and Computation*, pages 72–81. Springer, 1995.

10 R. Cole, M. Sharir, and C. K. Yap. On k-hulls and related problems. *SIAM J. Comput.*, 16:61–77, 1987.

11 G. B. Dantzig and M. N. Thapa. *Linear Programming 2: Theory and Extensions.* Springer, 2003.

12 R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Monographs in Computer Science. Springer, 1999.

13 H. Edelsbrunner and R. Waupotitsch. Computing a Ham-sandwich Cut in Two Dimensions. *J. Symb. Comput.*, 2(2):171–178, 1986.

14 H. G. Eggleston. *Convexity.* Cambridge University Press, 1963.

15 J. Erickson. New Lower Bounds for Convex Hull Problems in Odd Dimensions. *SIAM J. Comput.*, 28(4):1198–1214, 1999.

16 M. R. Fellows and N. Koblitz. Fixed-Parameter Complexity and Cryptography. *AAECC-10: Proceedings of the 10th International Symposium on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 121–131, 1993.

17 J. Flum and M. Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. Springer, 2006.

18 A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.*, 5(3):165–185, 1995.

19 P. Giannopoulos, C. Knauer, and G. Rote. The parameterized complexity of some geometric problems in unbounded dimension. In *Parameterized and exact computation. 4th international workshop, IWPEC 2009*, volume 5917 of *LNCS*, pages 198–209, 2009.

20 P. Giannopoulos, C. Knauer, M. Wahlström, and D. Werner. Hardness of discrepancy and related problems parameterized by the dimension. In *26th European Workshop on Computational Geometry (EuroCG)*, pages 173–176, 2010.

21 P. Giannopoulos, C. Knauer, and S. Whitesides. Parameterized Complexity of Geometric Problems. *Computer Journal*, 51(3):372–384, 2008.

22 E. Helly. Über Mengen konvexer Körper mit Gemeinschaftlichen Punkten. *Jahresbericht Deutsch. Math. Verein.*, 32:175–176, 1923.

23 R. Impagliazzo and R. Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

24 S. Jadhav and A. Mukhopadhyay. Computing a centerpoint of a finite planar set of points in linear time. In *SCG '93: Proceedings of the ninth annual symposium on Computational geometry*, pages 83–90. ACM, 1993.

25 C. Knauer. The complexity of geometric problems in high dimension. In *Theory and Applications of Models of Computation, 7th Annual Conference, TAMC 2010, Prague, Czech Republic, June 7-11, 2010. Proceedings*, pages 40–49, 2010.

26 S. Langerman and P. Morin. Covering Things with Things. *Discrete and Computational Geometry*, 33(4):717–729, 2005.

27 C.-Y. Lo, J. Matoušek, and W. Steiger. Ham-sandwich cuts in R^d. *STOC '92: Proceedings of the twenty-fourth annual ACM Symposium on Theory of Computing*, pages 539–545, 1992.

28 J. Matoušek. *Lectures on Discrete Geometry.* Springer, 2002.

29 J. Matoušek. *Using the Borsuk-Ulam Theorem - Lectures on Topological Methods in Combinatorics and Geometry.* Springer, 2003.

30 G. L. Miller and D. R. Sheehy. Approximate center points with proofs. In *SCG '09: Proceedings of the 25th annual symposium on Computational geometry*, pages 153–158. ACM, 2009.

31 N. Naor and M. Sharir. Computing a point in the center of a point set in three dimensions. In *CCCG: Canadian Conference in Computational Geometry*, 1990.

32 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, 2006.

33 C. H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.

34 M. Pătraşcu and R. Williams. On the Possibility of Faster SAT Algorithms. *Proc. 21st ACM/SIAM Symposium on Discrete Algorithms (SODA)*, 2010.

35 D.-H. Teng. *Points, spheres, and separators: A unified geometric approach to graph partitioning.* PhD thesis, School of Computer Science, Carnegie Mellon University, 1992.

36 F. Yu. On the Complexity of the Pancake Problem. *Electron. Notes Theor. Comput. Sci.*, 167:95–115, 2007.

37 G. M. Ziegler. *Lectures on Polytopes.* Springer, 1998.

Quantum query complexity of minor-closed graph properties*

Andrew M. Childs[1] and Robin Kothari[2]

1 Department of Combinatorics & Optimization
and Institute for Quantum Computing
University of Waterloo
amchilds@uwaterloo.ca

2 David R. Cheriton School of Computer Science
and Institute for Quantum Computing
University of Waterloo
rkothari@cs.uwaterloo.ca

Abstract

We study the quantum query complexity of minor-closed graph properties, which include such problems as determining whether an n-vertex graph is planar, is a forest, or does not contain a path of a given length. We show that most minor-closed properties—those that cannot be characterized by a finite set of forbidden subgraphs—have quantum query complexity $\Theta(n^{3/2})$. To establish this, we prove an adversary lower bound using a detailed analysis of the structure of minor-closed properties with respect to forbidden topological minors and forbidden subgraphs. On the other hand, we show that minor-closed properties (and more generally, sparse graph properties) that can be characterized by finitely many forbidden subgraphs can be solved strictly faster, in $o(n^{3/2})$ queries. Our algorithms are a novel application of the quantum walk search framework and give improved upper bounds for several subgraph-finding problems.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Quatum query complexity, quantum algorithms, lower bounds, graph minors, graph properties

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.661

1 Introduction

The decision tree model is a simple model of computation for which we can prove good upper and lower bounds. Informally, decision tree complexity, also known as query complexity, counts the number of input bits that must be examined by an algorithm to evaluate a function. In this paper, we focus on the query complexity of deciding whether a graph has a given property. The query complexity of graph properties has been studied for almost 40 years, yet old and easy-to-state conjectures regarding the deterministic and randomized query complexities of graph properties [11, 15, 20, 23] remain unresolved.

The study of query complexity has also been quite fruitful for quantum algorithms. For example, Grover's search algorithm [13] operates in the query model, and Shor's factoring algorithm [24] is based on the solution of a query problem. However, the quantum query complexity can be harder to pin down than its classical counterparts. For monotone graph properties, a wide class of graph properties including almost all the properties considered

* This work was supported in part by MITACS, NSERC, QuantumWorks, and the US ARO/DTO.

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 661–672
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

in this paper, the widely-believed Aanderaa–Karp–Rosenberg conjecture states that the deterministic and randomized query complexities are $\Theta(n^2)$, where n is the number of vertices. On the other hand, there exist monotone graph properties whose quantum query complexity is $\Theta(n)$, and others with query complexity $\Theta(n^2)$. In fact, one can construct a monotone graph property with quantum query complexity $\Theta(n^{1+\alpha})$ for any fixed $0 \leq \alpha \leq 1$ using known bounds for the threshold function [6].

The quantum query complexity of several specific graph properties has been established in prior work. Dürr, Heiligman, Høyer, and Mhalla [12] studied the query complexity of several graph problems, and showed in particular that connectivity has quantum query complexity $\Theta(n^{3/2})$. Zhang [28] showed that the quantum query complexity of bipartiteness is $\Theta(n^{3/2})$. Ambainis et al. [5] showed that planarity also has quantum query complexity $\Theta(n^{3/2})$. Berzina et al. [7] showed several quantum lower bounds on graph properties, including Hamiltonicity. Sun, Yao, and Zhang studied some non-monotone graph properties [26].

Despite this work, the quantum query complexity of many interesting graph properties remains unresolved. A well-studied graph property whose query complexity is unknown is the property of containing a triangle (i.e., a cycle on 3 vertices) as a subgraph. The triangle finding problem was first studied by Buhrman et al. [10], who gave an $O(n + \sqrt{nm})$ query algorithm for graphs with n vertices and m edges. With $m = \Theta(n^2)$, this approach uses $O(n^{3/2})$ queries, which matches the performance of the simple algorithm that searches for a triangle over the potential $\binom{n}{3}$ triplets of vertices. This was later improved by Magniez, Santha, and Szegedy [19] to $\tilde{O}(n^{1.3})$, and then by Magniez, Nayak, Roland, and Santha [18] to $O(n^{1.3})$, which is currently the best known algorithm. However, the best known lower bound for the triangle problem is only $\Omega(n)$ (by a simple reduction from the search problem). This is partly because one of the main lower bound techniques, the quantum adversary method of Ambainis [2], cannot prove a better lower bound due to the certificate complexity barrier [25, 28].

More generally, we can consider the H-subgraph containment problem, in which the task is to determine whether the input graph contains a fixed graph H as a subgraph. Magniez et al. also gave a general algorithm for H-subgraph containment using $\tilde{O}(n^{2-2/d})$ queries, where $d > 3$ is the number of vertices in H [19]. Again, the best lower bound known for H-subgraph containment is only $\Omega(n)$.

In this paper we study the quantum query complexity of minor-closed graph properties. A property is minor-closed if all minors of a graph possessing the property also possess the property. (Graph minors are defined in Section 2.) Since minor-closed properties can be characterized by forbidden minors, this can be viewed as a variant of subgraph containment in which we look for a given graph as a minor instead of as a subgraph. The canonical example of a minor-closed property is the property of being planar. Other examples include the property of being a forest, being embeddable on a fixed two-dimensional manifold, having treewidth at most k, or not containing a path of a given length.

While all minor-closed properties can be described by a finite set of forbidden minors, some minor-closed properties can also be described by a finite set of forbidden subgraphs, graphs that do not appear as a subgraph of any graph possessing the property. We call a graph property (which need not be minor closed) a *forbidden subgraph property* (FSP) if the property can be described by a finite set of forbidden subgraphs.

Our main result is that the quantum query complexity of minor-closed properties depends crucially on whether the property is FSP. In particular, Figure 1 summarizes our understanding of the quantum query complexity of minor-closed graph properties. All subgraph-closed properties, which include minor-closed properties and FSPs, have an easy lower bound

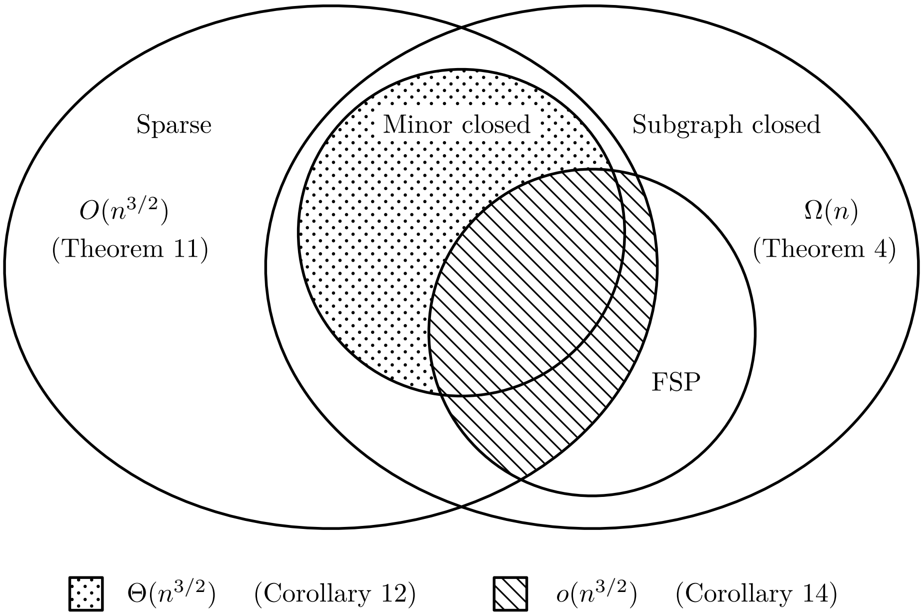


Figure 1 Summary of the main results.

of of $\Omega(n)$ (Theorem 4). Furthermore, all sparse graph properties, which are defined in Section 2 and which include all minor-closed properties, have an easy upper bound of $O(n^{3/2})$ (Theorem 11). On the lower bound side, our main contribution is to show that minor-closed properties that are not FSP require $\Omega(n^{3/2})$ queries (Theorem 8), which tightly characterizes their quantum query complexity. Regarding upper bounds, our main contribution is a quantum algorithm for all sparse graph properties that are FSP, using $O(n^{\alpha})$ queries for some $\alpha < 3/2$ that depends on the property (Corollary 14).

Our lower bounds (Section 3) use the quantum adversary method [2]. The basic idea of the lower bound is similar to the connectivity lower bound of Dürr et al. [12]. However, it is nontrivial to show that this approach applies using only the hypothesis that the property is minor-closed and not FSP. In fact, we show a slightly stronger result, assuming only that the property is not FSP and can be described by finitely many forbidden topological minors.

Our upper bounds (Section 4) use the quantum walk search formalism [18]. Our approach differs from previous applications of this formalism in several respects. We use several quantum walks occurring simultaneously on different Hamming graphs (whereas most previous quantum walk search algorithms used a single Johnson graph). Although this can be viewed as a single walk on a larger graph, the salient feature is that the walks on different graphs proceed at different speeds, i.e., in each time step a different number of steps are taken on each graph. In addition, we make essential use of the sparsity of the input graph.

By exploiting sparsity, we also improve upon known algorithms for many sparse graph properties, even if they are not necessarily minor closed. For example, we give improved algorithms for finding paths of a given length, as well as an algorithm that outperforms the general H-finding algorithm of Magniez et al. [19] whenever H is a bipartite graph.

Finally, as another application, we consider the C_4-subgraph containment problem. This can be viewed as a natural extension of the triangle problem, which is C_3-subgraph containment. Surprisingly, we show that C_4 finding can be solved with only $\tilde{O}(n^{1.25})$ queries, even faster than the best known upper bound for triangle finding, which is $O(n^{1.3})$.

2 Preliminaries

In this paper, all graphs are simple and undirected. Thus a graph on n vertices is specified by $\binom{n}{2}$ bits. In the query complexity model, the input graph is accessed by querying a black box to learn any of these $\binom{n}{2}$ bits. Deterministic and randomized algorithms have access to a black box taking two inputs, u and v, and returning a bit indicating whether (u, v) is an edge in the graph. To accommodate quantum algorithms, we define a quantum black box in the standard way. The quantum black box is a unitary gate that maps $|u, v, b\rangle$ to $|u, v, b \oplus e\rangle$ where $(u, v) \in V \times V$, b is a bit, and e is 1 if and only if $(u, v) \in E$.

Let the deterministic, randomized, and quantum query complexities of determining whether a graph possesses property $\mathcal{P}$ be denoted as $D(\mathcal{P})$, $R(\mathcal{P})$, and $Q(\mathcal{P})$, respectively, where for R and Q we consider two-sided bounded error. Clearly, $Q(\mathcal{P}) \le R(\mathcal{P}) \le D(\mathcal{P}) \le \binom{n}{2}$. Also note that these query complexities are the same for a property $\mathcal{P}$ and its complement $\bar{\mathcal{P}}$, since any algorithm for $\mathcal{P}$ can be turned into an algorithm for $\bar{\mathcal{P}}$ by negating the output, using no additional queries.

A graph property on n vertices is a property of n-vertex graphs that is independent of vertex labeling, i.e., isomorphic graphs are considered equivalent. For a graph G on n vertices and an n-vertex graph property $\mathcal{P}_n$, we write $G \in \mathcal{P}_n$ to mean that graph G has property $\mathcal{P}_n$. A graph property $\mathcal{P} := \{\mathcal{P}_n\}_{n=1}^{\infty}$ is a collection of n-vertex graph properties $\mathcal{P}_n$ for all $n \in N$. For example, the property "the first vertex is isolated" is not a graph property because it depends on the labeling, and in particular it depends on which vertex we decide to call the first one. However, the property "contains an isolated vertex" is a graph property.

An n-vertex graph property $\mathcal{P}_n$ is nontrivial if there exists a graph that possesses it and one that does not. A graph property $\mathcal{P} = \{\mathcal{P}_n\}_{n=1}^{\infty}$ is nontrivial if there exists an n_0 such that $\mathcal{P}_n$ is nontrivial for all $n > n_0$. Thus a property such as "contains a clique of size 5" is nontrivial, although it is trivial for graphs with fewer than 5 vertices.

In this paper, K_n and C_n refer to the complete graph and cycle on n vertices, respectively. $K_{s,t}$ is the complete bipartite graph with s vertices in one part and t vertices in the other. A d-path is a path with d edges (i.e., with $d + 1$ vertices). For a graph G, $V(G)$ and $E(G)$ denote the vertex and edge sets of the graph; $n := |V(G)|$ and $m := |E(G)|$.

A graph H is said to be a subgraph of G, denoted $H \le_S G$, if H can be obtained from G by deleting edges and isolated vertices. A graph H is said to be a minor of G, denoted $H \le_M G$, if H can be obtained from G by deleting edges, deleting isolated vertices, and contracting edges. To contract an edge (u, v), we delete the vertices u and v (and all associated edges) and create a new vertex that is adjacent to all the original neighbors of u and v. The name "edge contraction" comes from viewing this operation as shrinking the edge (u, v) to a point, letting the vertices u and v coalesce to form a single vertex.

Another way to understand graph minors is to consider reverse operations: $H \le_M G$ if G can be obtained from H by adding isolated vertices, adding edges, and performing vertex splits. In a vertex split, we delete a vertex u and add two new adjacent vertices v and w, such that each original neighbor of u becomes a neighbor of either v or w, or both. In general, this operation does not lead to a unique graph, since there may be many different ways to split a vertex.

A related operation, which is a special case of a vertex split, is known as an elementary subdivision. This operation replaces an edge (u, v) with two edges (u, w) and (w, v), where w is a new vertex. A graph H is said to be a topological minor of G, denoted $H \le_T G$, if G can be obtained from H by adding edges, adding isolated vertices, and performing elementary subdivisions. We call G a subdivision of H if it is obtained from H by performing any number of elementary subdivisions.

Some graph properties can be expressed using a forbidden graph characterization. Such a characterization says that graphs have the property if and only if they do not contain any of some set of forbidden graphs according to some notion of graph inclusion, such as subgraphs or minors. For example, a graph is a forest if and only if it contains no cycle as a subgraph, so forests are characterized by the forbidden subgraph set $\{C_k : k \geq 3,\ k \in \mathbb{N}\}$. The property of being a forest can also be characterized by the single forbidden minor C_3, since a graph is a forest if and only if it does not contain C_3 as a minor. If a property can be expressed using a finite number of forbidden subgraphs, we call it a forbidden subgraph property (FSP). A property is said to be subgraph closed if every subgraph of a graph possessing the property also possess the property. Similarly, a property is said to be minor closed if all minors of a graph possessing the property also possess the property. In a series of 20 papers spanning over 20 years, Robertson and Seymour proved the following theorem [22]:

▶ **Theorem 1** (Graph minor theorem). *Every minor-closed graph property can be described by a finite set of forbidden minors.*

We also require the following consequence of the graph minor theorem, which follows using well-known facts about topological minors [21, Theorem 2.1].

▶ **Corollary 2.** *Every minor-closed graph property can be described by a finite set of forbidden topological minors.*

We call a graph property *sparse* if there exists a constant c such that every graph G with the property has $|E(G)| \leq c\,|V(G)|$. Nontrivial minor-closed properties are sparse, which is an easy corollary of Mader's theorem [17].

▶ **Theorem 3.** *Every nontrivial minor-closed graph property is sparse.*

We use $\tilde{O}$ notation to denote asymptotic upper bounds that neglect logarithmic factors. Specifically, $f(n) = \tilde{O}(g(n))$ means $f(n) = O(g(n) \log^k g(n))$ for some constant k.

3 Lower bounds

The following lower bound follows easily using the adversary methods of Ambainis [2] and Aaronson [1].

▶ **Theorem 4.** *For any nontrivial subgraph-closed graph property $\mathcal{P}$, $Q(\mathcal{P}) = \Omega(n)$, $R(\mathcal{P}) = \Theta(n^2)$, and $D(\mathcal{P}) = \Theta(n^2)$.*

With the exception of general sparse properties, this theorem covers all the properties considered in this paper, since every property (or its complement) is closed under subgraphs. Thus all the properties considered in this paper are classically uninteresting from the viewpoint of query complexity, since their classical (deterministic or randomized) query complexity is exactly $\Theta(n^2)$.

In the remainder of this section, we describe our main lower bound result: Every minor-closed property that is not FSP has $Q(\mathcal{P}) = \Omega(n^{3/2})$. We begin by considering H-topological minor containment properties that are not also H-subgraph containment properties, while describing some of the tools used to show the more general result.

As a motivating example, consider $H = C_3$. C_3-topological minor containment (which is equivalent to C_3-minor containment) is the property of being cyclic; its complementary property is that of being a forest. We show that $\Omega(n^{3/2})$ queries are required for this property.

This lower bound is similar to the connectivity lower bound of Dürr et al. [12]. The intuition is that a long path and a long cycle look the same locally. Since algorithms only have access to local information, these two graphs should be hard to distinguish. Unfortunately this is not sufficient, since a path can be easily distinguished from a cycle by searching the entire graph for a degree-1 vertex, which can be done with $O(n)$ queries. Instead, we try to distinguish a path from the disjoint union of a cycle and a path. Now both graphs have 2 degree-1 vertices. We require both the cycle and the path to be long, since a short cycle or path could be quickly traversed. Considering these instances, an adversary argument shows the following.

▶ **Theorem 5.** *Deciding if a graph is a forest requires $\Omega(n^{3/2})$ queries.*

This construction does not use any particular property of forests, except that all subdivisions of C_3 are not forests, and that if we delete an edge from a subdivision of C_3, the resulting graph is a forest. More precisely, we use the existence of a graph G (in this case C_3) and an edge $(u, v) \in E(G)$ (in this case it can be any edge) such that if (u, v) is subdivided any number of times, the resulting graph still does not have the property (in this case, of being a forest) and if (u, v) is replaced by two disjoint paths the resulting graph does have the property. The following lemma formalizes this intuition.

▶ **Lemma 6.** *Let $\mathcal{P}$ be a graph property closed under topological minors. If there exists a graph $G \notin \mathcal{P}$ and an edge (u, v) of G, such that replacing the edge (u, v) by two disjoint paths of any length, one connected to vertex u and the other connected to vertex v, always results in a graph $G' \in \mathcal{P}$, then $Q(\mathcal{P}) = \Omega(n^{3/2})$.*

It can be shown that any graph H for which H-topological minor containment does not coincide with H-subgraph containment contains such an edge, and thus we obtain an $\Omega(n^{3/2})$ lower bound in this case. In particular, a graph H satisfies this condition if and only if it is cyclic or contains 2 vertices of degree at least 3 in the same connected component, and any edge on a cycle or on a path between 2 vertices of degree at least 3 can serve as the edge (u, v) in Lemma 6.

From H-topological minor containment, we move on to properties that can be described by a finite set of forbidden topological minors. While this case is conceptually similar to the case of a single forbidden minor, it is more technically challenging. The final result, however, is easy to state:

▶ **Lemma 7.** *For any graph property $\mathcal{P}$ that is not FSP and that is described by a finite set of forbidden topological minors, there exists a graph $G \notin \mathcal{P}$ and an edge $(u, v) \in E(G)$ satisfying the conditions of Lemma 6.*

Combining this with Corollary 2 and Lemma 6, we get our main lower bound result.

▶ **Theorem 8.** *For any nontrivial minor-closed property $\mathcal{P}$ that is not FSP, $Q(\mathcal{P}) = \Omega(n^{3/2})$.*

This lower bound cannot be improved due to a matching algorithm shown in Section 4. It cannot be extended to minor-closed properties that are also FSP because, as we also show in Section 4, every property of this type has query complexity $o(n^{3/2})$.

4 Algorithms

We now turn to quantum algorithms for deciding minor-closed graph properties, as well as related algorithms for subgraph-finding problems.

4.1 Sparse graph detection and extraction

We begin by describing some basic tools that allow us to detect whether a graph is sparse and to optimally extract the adjacency matrix of a sparse graph.

To tell whether a graph is sparse, we can apply quantum counting to determine approximately how many edges it contains. In particular, Theorem 15 of [9] can be applied to show the following.

▶ **Lemma 9.** *For any constant $\epsilon > 0$ and function $f : \mathbb{Z}^+ \to \mathbb{Z}^+$ there is a quantum algorithm using $O(\sqrt{n^2/f(n)} \log \frac{1}{\delta})$ queries that accepts graphs with $m \geq (1+\epsilon)f(n)$ and rejects graphs with $m \leq (1-\epsilon)f(n)$ with probability at least $1-\delta$.*

We also use a procedure for extracting all marked items in a search problem.

▶ **Lemma 10.** *Let $f : \{1, \dots, N\} \to \{0, 1\}$ be a black-box function with $|f^{-1}(1)| = K$. The bounded-error quantum query complexity of determining $f^{-1}(1)$ is $O(\sqrt{NK})$ if $K > 0$, and $O(\sqrt{N})$ if $K = 0$.*

This result and its optimality appear to be folklore (see for example [4]).

An easy consequence of these results is that sparse graph properties can be decided in $O(n^{3/2})$ queries.

▶ **Theorem 11.** *If $\mathcal{P}$ is a sparse graph property, then $Q(\mathcal{P}) = O(n^{3/2})$.*

Combining this with Theorem 3 and Theorem 8, an immediate consequence is

▶ **Corollary 12.** *If $\mathcal{P}$ is nontrivial, minor closed, and not FSP, then $Q(\mathcal{P}) = \Theta(n^{3/2})$.*

Note that this provides an alternative proof that the quantum query complexity of planarity is $\Theta(n^{3/2})$ [5].

For minor-closed graph properties that are also FSP, the lower bounds from Section 3 do not rule out the possibility of an improvement over Theorem 11. In fact, we show that an improvement is possible for all such properties.

4.2 Quantum walk search

Our algorithms use the quantum walk search framework of Magniez et al. (Theorem 3 of [18]), which builds on the work of Ambainis [3] and Szegedy [27]. However, our application of this framework differs from previous applications in several ways.

In nearly all previous quantum walk search algorithms, the graph on which the walk occurs is the Johnson graph $J(N, K)$, whose vertices are the $\binom{N}{K}$ subsets of $\{1, \dots, N\}$ of size K, with an edge between subsets that differ in exactly one item. The parameters N and K are chosen based on the input size. For example, the triangle finding algorithm [19] fixes $N = n$ and $K = n^{3/5}$, where n is the number of vertices in the input graph (not to be confused with the graph on which the quantum walk occurs). Each vertex of the Johnson graph has an associated data structure. Populating this data structure typically requires queries to the input. In the setup step, the walk begins in a uniform superposition over all the vertices of the Johnson graph. A step of the quantum walk corresponds to moving to one of the neighbors of the current vertex and updating the data structure; this is called the update step. Some of the vertices are designated as “marked” and the objective of the walk is to determine if there are any such vertices. Every few steps of the walk, we have a checking step which determines if the current vertex is marked.

Our walk differs from this in several ways. For our purposes it will be more convenient to replace the Johnson graph $J(N, K)$ by the Hamming graph $H(N, K)$, with vertex set $\{1, \ldots, N\}^K$ and edges between two K-tuples that differ in exactly one coordinate. This choice simplifies the implementation of our setup step. Although the order of the items has no significance, and the possibility of repeated items only slows down the algorithm, the effect is not significant.

Furthermore, our algorithms involve a walk on several different Hamming graphs. Each Hamming graph may have very different value of N and K, and these values depend not only on the input size, but on the actual input itself. This means that queries are required even to decide which graph is being walked on. Moreover, the walks on different graphs may occur at different speeds.

4.3 Detecting subgraphs of sparse graphs

We now describe algorithms that determine whether a sparse graph G contains a given subgraph H. We begin with an informal overview before stating the general result.

For concreteness, consider the problem of detecting a clique of size 5 in a sparse graph. The idea is to look for the 5 vertices separately (using 5 quantum walks), and use the checking step to verify that the 5 vertices do form a clique. The data structure stores a list of the neighbors of each vertex under consideration.

First, let us assume that a clique of size 5 exists; if our algorithm fails to detect one we can conclude that our assumption was incorrect and reject the input. Second, we guess the approximate degrees of all the vertices we are looking for, up to a constant multiplicative factor. Since $\Theta(\log n)$ guesses cover the entire range between 1 and n, up to a multiplicative overhead of $\mathrm{poly}(\log n)$, we can assume that we already know the approximate degrees of all the vertices we are looking for. Let these degrees be q_i for $1 \leq i \leq 5$.

Then we use quantum counting to estimate the number of vertices with approximate degree q_i. Let this value be t_i. Now we set up 5 quantum walks, with the i^{th} walk searching over k_i-tuples of the t_i vertices of degree near q_i. To optimize the overall walk, we take α_i steps for the i^{th} component, with carefully chosen values of the k_is and the α_is. Since we store the list of neighbors for each vertex, no queries are needed to check whether a set of 5 vertices forms a clique; queries are used only in the setup step (preparing the initial superposition) and the update step (taking α_i steps on the i^{th} Hamming graph for each i).

Storing the list of neighbors is costly for high-degree vertices, but cheap for low-degree vertices. However, since we know the graph is sparse, there cannot be too many high-degree vertices, so we are able to accept a high cost for such vertices. On the other hand, there may be many low-degree vertices, but since they are cheap to process, the total cost associated with all the vertices of a given approximate degree is about the same. This is why we break up the search space by grouping together vertices of similar degree. Note that the sparsity of the input graph is essential to upper bound the number of high-degree vertices in the graph.

Finally, note that it was unnecessary to search for all 5 vertices of the clique. Since we store the list of neighbors of each vertex, we can instead search for only 4 vertices, using the neighbor lists to determine whether they share a common neighbor. This idea naturally generalizes to storing a *vertex cover* of the graph H we are looking for, a subset C of the vertices of H such that each edge of H involves at least one vertex from C. Given neighbor lists for any particular subset of vertices, we can determine whether that subset includes a vertex cover of H with no further queries.

Our general strategy is to search over tuples of the vertices of G for one containing a vertex cover of H. We exploit sparsity by separately considering cases where the vertices of the vertex cover have given (approximate) degrees. Letting $\mathrm{vc}(H)$ denote the smallest number of vertices in any vertex cover of H, we have the following result.

▶ **Theorem 13.** *Let $\mathcal{P}$ be the property that a graph either has more than cn edges (for some constant c) or contains a given subgraph H. Then $Q(\mathcal{P}) = \tilde{O}\left(n^{\frac{3}{2}-\frac{1}{\mathrm{vc}(H)+1}}\right)$.*

The analysis proceeds roughly as follows. First we reject non-sparse graphs using Lemma 9. Then the quantum walk search algorithm uses $O(S + \frac{1}{\sqrt{\epsilon}}(\frac{1}{\sqrt{\delta}}U + C))$ queries, where S, U, C are the setup, update, and checking costs, respectively; ϵ is the fraction of marked states; and δ is the spectral gap of the walk [18]. We have $S = O(\sum_i k_i n/\sqrt{t_i})$, $U = O(\sum_i \alpha_i n/\sqrt{t_i})$, and $C = 0$. Furthermore, $\epsilon = \Omega(\prod_i k_i/t_i)$ and $\delta = \Omega(\min_i \alpha_i/k_i)$. By choosing $\alpha_i/\alpha_j = k_i/k_j = \sqrt{t_i/t_j}$, $\alpha_1 = 1$ (assuming without loss of generality that this is the smallest α_i), and $k_1 = \sqrt{t_1}n^{\frac{1}{2}-\frac{1}{\mathrm{vc}(H)+1}}$, we obtain the stated running time. Various technical issues that arise only increase the running time by a polylogarithmic factor.

We can apply this algorithm to decide sparse graph properties, and in particular minor-closed properties, that are also FSP: we simply search for each of the forbidden subgraphs, accepting if none of them are present. For minor-closed properties, the non-sparseness condition of Theorem 13 can be removed due to Theorem 3. Thus, since $\mathrm{vc}(H)$ is a constant for any fixed graph H, we have the following.

▶ **Corollary 14.** *If $\mathcal{P}$ is sparse and FSP, then $Q(\mathcal{P}) = o(n^{3/2})$.*

For many subgraphs, we can improve Theorem 13 further by storing more information about the vertices in the vertex cover: in addition to storing their neighborhoods, we can store basic information about their second neighbors. In particular, we have the following.

▶ **Theorem 15.** *Let $\mathcal{P}$ be the property that a graph either has more than cn edges (for some constant c) or contains a given subgraph H. Let H' be the graph obtained by deleting all degree-one vertices of H that are not part of an isolated edge. Then $Q(\mathcal{P}) = \tilde{O}\left(n^{\frac{3}{2}-\frac{1}{\mathrm{vc}(H')+1}}\right)$.*

Theorem 15 gives an improvement over Theorem 13 for properties that are characterized by a single forbidden minor (and equivalently, a single forbidden subgraph). For example, we have the following.

▶ **Theorem 16.** *A d-path with $d \geq 3$ can be detected using $\tilde{O}(n^{\frac{3}{2}-\frac{1}{\lceil d/2 \rceil}})$ quantum queries. In particular, the quantum query complexity of detecting a d-path for $d \in \{1,2,3,4\}$ is $\tilde{\Theta}(n)$.*

Note that even the improved result from Theorem 15 has zero checking cost. We can sometimes obtain a further improvement by performing nontrivial checking. For example, we can detect 7-paths in the same complexity that Theorem 16 gives for 5- and 6-paths and we can detect 9- and 10-paths in the same complexity that Theorem 16 gives for 7- and 8-paths:

▶ **Theorem 17.** *H-subgraph containment has query complexity $\tilde{O}(n^{7/6})$ if H is a 7-path and $\tilde{O}(n^{5/4})$ if H is a 9- or 10-path.*

Similar improvements are also possible for longer paths; we omit the details here.

4.4 Relaxing sparsity

So far we have focused on sparse graphs, since this is the relevant case for minor-closed properties. However, our algorithms easily generalize to the case where the number of edges is at most any prescribed upper bound, leading to further applications.

▶ **Theorem 18.** *Let $\mathcal{P}$ be the property that an n-vertex graph has at most $\bar{m}$ edges (where $\bar{m} = \Omega(n)$) and contains a given subgraph H. Let H' be the graph obtained by deleting all degree-one vertices of H that are not part of an isolated edge. Then $Q(\mathcal{P}) = \tilde{O}(\sqrt{\bar{m}}\, n^{1-\frac{1}{\mathrm{vc}(H')+1}})$.*

In conjunction with the Kővári-Sós-Turán theorem [16], this algorithm has applications to subgraph-finding problems that are not equivalent to minor-finding problems.

▶ **Theorem 19** (Kővári-Sós-Turán)**.** *If a graph G on n vertices does not contain $K_{s,t}$ as a subgraph, where $1 \le s \le t$, then $|E(G)| \le c_{s,t}\, n^{2-\frac{1}{s}}$, where $c_{s,t}$ is a constant depending only on s and t.*

Suppose H is a d-vertex bipartite graph. Theorem 19 shows that if $|E(G)| > c\, n^{2-\frac{2}{d}}$ (for some constant c), then G must contain $K_{s,d-s}$ for all $1 \le s \le d/2$, and in particular, must contain H. Combining this with the fact that $\mathrm{vc}(H') \le \mathrm{vc}(H) \le d/2$, we have the following.

▶ **Theorem 20.** *If H is a d-vertex bipartite graph, then H-subgraph containment has quantum query complexity $\tilde{O}(n^{2-\frac{1}{d}-\frac{2}{d+2}}) = \tilde{O}(n^{2-\frac{3d+2}{d(d+2)}})$.*

Recall that for $d > 3$, Theorem 4.6 of [19] gives an upper bound of $\tilde{O}(n^{2-\frac{2}{d}})$ for finding a d-vertex subgraph. For bipartite subgraphs, Theorem 20 is a strict improvement.

Note that a better bound may be possible by taking the structure of H into account. In general, if H is a bipartite graph with the i^{th} connected component having vertex bipartition $V_i \cup U_i$ with $1 \le |V_i| \le |U_i|$, then we can replace $d/2$ by $\sum_i |V_i|$, since a graph that contains $K_{\sum_i |V_i|, \sum_i |U_i|}$ must contain H, and $\mathrm{vc}(H') \le \mathrm{vc}(H) = \sum_i |V_i|$. As a simple example, if $H = K_{1,t}$ is a star on $t+1$ vertices, then H-subgraph containment can be solved with $\tilde{O}(n)$ quantum queries (which is essentially optimal due to Theorem 4).

Just as mentioned at the end of Section 4.3, we can sometimes improve over Theorem 18 by introducing a nontrivial checking cost. The following is a simple example of such an algorithm, using a result on the sparsity of graphs that exclude C_4 [8].

▶ **Theorem 21.** *C_4-subgraph containment can be solved in $\tilde{O}(n^{1.25})$ quantum queries.*

This may seem unexpected, since C_4 finding is a natural generalization of triangle finding to a larger subgraph. Indeed, the previous best known quantum algorithm for C_4 finding used $\tilde{O}(n^{1.5})$ queries [19], more than the $O(n^{1.3})$ queries for triangle finding. Our improvement shows that 4-cycles can be found in fewer quantum queries than in the best known quantum algorithm for finding 3-cycles.

5 Conclusions and open problems

In this paper, we have studied the quantum query complexity of minor-closed graph properties. The difficulty of such problems depends crucially on whether the property can also be characterized by a finite set of forbidden subgraphs. Minor-closed properties that are not characterized by forbidden subgraphs have matching upper and lower bounds of $\Theta(n^{3/2})$ (Corollary 12), whereas all minor-closed properties that can be expressed in terms of forbidden subgraphs can be solved strictly faster, in $o(n^{3/2})$ queries (Corollary 14).

Since the best known lower bound for the latter class of problems is the simple $\Omega(n)$ lower bound from Theorem 4, an obvious open question is to give improved upper or lower bounds for subgraph-finding problems. While the standard quantum adversary method cannot prove a better lower bound, it might be possible to apply the negative weights adversary method [14] or the polynomial method [6]. Note that sparsity makes forbidden subgraph properties potentially more difficult to lower bound; this is precisely the feature we took advantage of in the algorithms of Section 4. Proving a superlinear lower bound for any subgraph-finding problem—even one for which dense graphs might not contain the subgraph, such as in the case of triangles—remains a major challenge. On the algorithmic side, note that while our algorithms take advantage of sparsity, minor-closed families of graphs have other special properties, such as bounded degeneracy, that might also be exploited.

The algorithms described in Section 4 have several features not shared by previous quantum walk search algorithms for graph properties: queries are required even to identify which vertices of the input graph to search over (namely, to find vertices of a certain degree), and the performance of the walk is optimized by making different transitions at different rates. We hope these techniques might prove useful in other quantum algorithms.

Note that Theorem 13 can be applied to find induced subgraphs (just as with the algorithms of [19]). However, the improvements described in Theorem 15 and Theorem 18 do not apply to induced subgraphs, and in general it could be easier or more difficult to decide whether a given graph is present as an induced subgraph rather than a (not necessarily induced) subgraph. It might be fruitful to explore induced subgraph finding more generally.

It might also be interesting to focus on finding natural families of subgraphs such as paths. Recall that we showed the quantum complexity of this problem is $\tilde{\Theta}(n)$ for lengths up to 4 and $\tilde{O}(n^{7/6})$ for lengths of 5, 6, and 7, with nontrivial algorithms for longer paths as well (Theorem 16 and Theorem 17). The case of paths of length 5, the smallest case for which our algorithm is not known to be optimal, appears to be a natural target for future work.

Acknowledgements We thank Frédéric Magniez for suggesting that a strategy similar to the lower bound for connectivity in [12] could be applied to C_3-minor finding. R. K. thanks Jim Geelen for helpful discussions about graph minors, and Vinayak Pathak and Hrushikesh Tilak for interesting conversations about many aspects of this work.

References

1 S. Aaronson. Lower bounds for local search by quantum arguments. *SIAM Journal on Computing*, 35(4):804–824, 2006. Preliminary version in STOC 2004.

2 A. Ambainis. Quantum lower bounds by quantum arguments. *Journal of Computer and System Sciences*, 64(4):750–767, 2002. Preliminary version in STOC 2000.

3 A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. Preliminary version in FOCS 2004.

4 A. Ambainis. A new quantum lower bound method, with an application to strong direct product theorem for quantum search. *Theory of Computing*, 6(1):1–25, 2010.

5 A. Ambainis, K. Iwama, M. Nakanishi, H. Nishimura, R. Raymond, S. Tani, and S. Yamashita. Quantum query complexity of boolean functions with small on-sets. In *Proceedings of the 19th International Symposium on Algorithms and Computation*, pages 907–918, 2008.

6 R. Beals, H. Buhrman, R. Cleve, M. Mosca, and R. de Wolf. Quantum lower bounds by polynomials. *Journal of the ACM*, 48(4):778–797, 2001. Preliminary version in FOCS 1998.

7 A. Berzina, A. Dubrovsky, R. Freivalds, L. Lace, and O. Scegulnaja. Quantum query complexity for some graph problems. In *SOFSEM 2004: Theory and Practice of Computer Science*, volume 2932, pages 1–11, 2004.

8 J. A. Bondy and M. Simonovits. Cycles of even length in graphs. *Journal of Combinatorial Theory, Series B*, 16(2):97–105, 1974.

9 G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. In *Quantum computation and information*, volume 305 of *Contemporary Mathematics*, pages 53–74. AMS, 2002.

10 H. Buhrman, C. Dürr, M. Heiligman, P. Høyer, F. Magniez, M. Santha, and R. de Wolf. Quantum algorithms for element distinctness. *SIAM Journal on Computing*, 34(6):1324–1330, 2005. Preliminary version in CCC 2000.

11 A. Chakrabarti and S. Khot. Improved lower bounds on the randomized complexity of graph properties. *Random Structures and Algorithms*, 30(3):427–440, 2007. Preliminary version in ICALP 2001.

12 C. Dürr, M. Heiligman, P. Høyer, and M. Mhalla. Quantum query complexity of some graph problems. *SIAM Journal on Computing*, 35(6):1310–1328, 2006.

13 L. K. Grover. Quantum mechanics helps in searching for a needle in a haystack. *Physical Review Letters*, 79(2):325–328, 1997. Preliminary version in STOC 1996.

14 P. Høyer, T. Lee, and R. Spalek. Negative weights make adversaries stronger. In *STOC '07: Proceedings of the 39th Annual ACM Symposium on Theory of Computing*, pages 526–535, 2007.

15 J. Kahn, M. Saks, and D. Sturtevant. A topological approach to evasiveness. In *SFCS '83: Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 31–33, 1983.

16 T. Kövari, V. T. Sós, and P. Turán. On a problem of K. Zarankiewicz. *Colloquium Mathematicum*, 3:50–57, 1954.

17 W. Mader. Homomorphieeigenschaften und mittlere Kantendichte von Graphen. *Mathematische Annalen*, 174:265–268, 1967.

18 F. Magniez, A. Nayak, J. Roland, and M. Santha. Search via quantum walk. In *STOC '07: Proceedings of the 39th ACM Symposium on Theory of Computing*, pages 575–584, 2007.

19 F. Magniez, M. Santha, and M. Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–424, 2007. Preliminary version in SODA 2005.

20 R. L. Rivest and J. Vuillemin. A generalization and proof of the Aanderaa-Rosenberg conjecture. In *STOC '75: Proceedings of the 7th Annual ACM Symposium on Theory of Computing*, pages 6–11, 1975.

21 N. Robertson and P. D. Seymour. Graph minors. VIII. A Kuratowski theorem for general surfaces. *Journal of Combinatorial Theory, Series B*, 48(2):255–288, 1990.

22 N. Robertson and P. D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325–357, 2004.

23 A. L. Rosenberg. On the time required to recognize properties of graphs: a problem. *SIGACT News*, 5(4):15–16, 1973.

24 P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. Preliminary version in FOCS 1994.

25 R. Špalek and M. Szegedy. All quantum adversary methods are equivalent. *Theory of Computing*, 2(1):1–18, 2006. Preliminary version in ICALP 2005.

26 X. Sun, A. C. Yao, and S. Zhang. Graph properties and circular functions: How low can quantum query complexity go? In *CCC '04: Proceedings of the 19th Annual IEEE Conference on Computational Complexity*, pages 286–293, 2004.

27 M. Szegedy. Quantum speed-up of Markov chain based algorithms. In *FOCS '04: Proceedings of the 45th IEEE Symposium on Foundations of Computer Science*, pages 32–41, 2004.

28 S. Zhang. On the power of Ambainis lower bounds. *Theoretical Computer Science*, 339(2):241–256, 2005. Preliminary version in ICALP 2004.

Three Query Locally Decodable Codes with Higher Correctness Require Exponential Length

Anna Gál* **and Andrew Mills**

Department of Computer Science
University of Texas at Austin
panni@cs.utexas.edu
amills@cs.utexas.edu

Abstract

Locally decodable codes are error correcting codes with the extra property that, in order to retrieve the correct value of just one position of the input with high probability, it is sufficient to read a small number of positions of the corresponding, possibly corrupted codeword. A breakthrough result by Yekhanin showed that 3-query linear locally decodable codes may have subexponential length.

The construction of Yekhanin, and the three query constructions that followed, achieve correctness only up to a certain limit which is $1 - 3\delta$ for nonbinary codes, where an adversary is allowed to corrupt up to δ fraction of the codeword. The largest correctness for a subexponential length 3-query binary code is achieved in a construction by Woodruff, and it is below $1 - 3\delta$.

We show that achieving slightly larger correctness (as a function of δ) requires exponential codeword length for 3-query codes. Previously, there were no larger than quadratic lower bounds known for locally decodable codes with more than 2 queries, even in the case of 3-query linear codes. Our results hold for linear codes over arbitrary finite fields and for binary nonlinear codes.

Considering larger number of queries, we obtain lower bounds for q-query codes for $q > 3$, under certain assumptions on the decoding algorithm that have been commonly used in previous constructions. We also prove bounds on the largest correctness achievable by these decoding algorithms, regardless of the length of the code. Our results explain the limitations on correctness in previous constructions using such decoding algorithms. In addition, our results imply tradeoffs on the parameters of error correcting data structures.

Digital Object Identifier 10.4230/LIPIcs.STACS.2011.673

1 Introduction

Locally decodable codes are error correcting codes with the extra property that, in order to retrieve the correct value of just one position of the input with high probability, it is sufficient to read a sublinear or even just a constant number of positions of the corresponding, possibly corrupted, codeword. The formal definition was given by Katz and Trevisan [9] in 2000.

▶ **Definition 1.1.** (Katz and Trevisan [9]) For reals δ and ϵ, and a natural number q, we say that $\mathbf{C}\colon \Sigma^n \to \Gamma^m$ is a (q, δ, ϵ)-*Locally Decodable Code (LDC)* if there exists a probabilistic algorithm A such that: in every invocation, A reads at most q positions of y; and for every $x \in \Sigma^n$ and $y \in \Gamma^m$ with $d(y, \mathbf{C}(x)) \le \delta m$, and for every $i \in [n]$, we have $\Pr[A^y(i) = x_i] \ge \frac{1}{|\Sigma|} + \epsilon$, where the probability is taken over the internal coin tosses of A.

We will refer to the value $\frac{1}{|\Sigma|} + \epsilon$ in Definition 1.1 as the *correctness* of the given decoding algorithm A, while ϵ can be thought of as the *advantage* over random guessing.

[*] Supported in part by NSF Grants CCF-0830756 and CCF-1018060

28th Symposium on Theoretical Aspects of Computer Science (STACS'11).
Editors: Thomas Schwentick, Christoph Dürr; pp. 673–684
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

Locally decodable codes have interesting applications, both in complexity theory and in practical areas. Locally decodable codes are especially useful in situations where we want to encode large amounts of data to protect against errors, but need to be able to access individual units; for example, individual patient records of a large hospital. Encoding each unit separately would give less protection against errors, and encoding the whole data set with a traditional error correcting code would require reading the whole encoded database just to access small parts of it. Locally decodable codes are closely related to private information retrieval: constructions of good locally decodable codes yield efficient protocols for private information retrieval. Private information retrieval schemes allow users to retrieve information from databases without revealing information about which data items the user is retrieving. Other applications and related structures include self correcting computations, random self-reducibility, probabilistically checkable proofs. See [15] for a survey. More recently, [6] related LDCs to polynomial identity testing for arithmetic circuits, and [4] to matrix rigidity and circuit lower bounds.

It is quite remarkable that such codes exist at all for constant number of queries. A simple example is the Hadamard code, which has the property that any input bit can be recovered with probability at least $1 - 2\delta$ from codewords possibly corrupted in up to δm positions, by a randomized algorithm that in every invocation reads no more than 2 bits of the code. However, the code is very large: the length of the codewords is 2^n for encoding n bit inputs.

Of course it would be desirable to have much more efficient, in particular polynomial length codes, but this seems to be currently out of reach for constant number of queries. Efficient constructions are known for large number of queries. See [15] for a survey.

It is known that for large enough n, 1-query locally decodable codes (that read at most one bit) cannot do better than random guessing [9].

For 2-query linear codes essentially tight bounds are known: Goldreich, Karloff, Schulman and Trevisan [8] proved exponential lower bounds for 2-query linear codes over finite fields up to a certain field size. This was later extended by Dvir and Shpilka [6] to give exponential lower bounds for 2-query linear codes over arbitrary fields. Further improvements for the 2-query linear case were given by [12, 14]. Shiowattana and Lokam [14] prove a lower bound of $\Omega(2^{4\delta n/(1-2\epsilon)})$, which is tight within a constant factor, for 2-query binary linear locally decodable codes.

Kerenidis and de Wolf [10] proved exponential lower bounds for arbitrary binary (not necessarily linear) 2-query locally decodable codes, based on quantum arguments. They also extended their lower bounds to codes over larger alphabets, but the bound decreases with the alphabet size. The strongest lower bounds so far for nonlinear codes from $\{0,1\}^n$ to $\Sigma^m = (\{0,1\}^\ell)^m$ were proved by Wehner and de Wolf [16], and are of the form $2^{\Omega(\delta\epsilon^2 n/(2^{2\ell}))}$. A proof of the 2-query lower bound for binary codes is given in [3] without using quantum arguments. It is still open to obtain nontrivial lower bounds for 2-query nonlinear codes over alphabets of size $\Omega(\sqrt{n})$.

For larger number of queries, there is still a huge gap between the known upper and lower bounds, even for binary linear codes. For codes over small (constant size) alphabets Katz and Trevisan in [9] gave a general lower bound that holds for any q showing that q-query locally decodable codes must have length $\Omega(n^{q/(q-1)})$. This bound was slightly improved by Kerenidis and de Wolf [10] to $\Omega((n/\log n)^{\frac{q+1}{q-1}})$, and by Woodruff [18] to $\Omega(n^{\frac{q+1}{q-1}})/\log n$. Woodruff [20] proved $\Omega(n^2)$ lower bounds on the length of 3-query linear codes over any field. Prior to our work, no larger than n^2 lower bound was known for locally decodable codes that allow more than 2 queries, even in the case of 3-query linear codes.

A breakthrough result of Yekhanin [21] showed that subexponential length 3-query linear locally decodable codes exist, under assumptions about the existence of infinitely many Mersenne primes. Raghavendra [13] gave some simplifications to Yekhanin's codes. Building on these works, Efremenko [7] gave a construction of subexponential length 3-query linear locally decodable codes without any unproven assumptions. All these constructions have a limit on the correctness achieved by the algorithm as a function of δ where an adversary can corrupt up to δ fraction of the codeword positions. Efremenko's construction [7] gives $1-3\delta$ correctness for a 3-query nonbinary code. For 3-query binary codes, the best dependence between the parameters is achieved in a paper by Woodruff [19], which yields 3-query binary linear locally decodable codes with correctness close to, but still below, $1 - 3\delta$. Note that these results do not provide correctness larger than $1/2$, that is they do not give better correctness than random guessing for binary codes, if the fraction of corrupted positions δ is larger than $1/6$. Recent results of Ben-Aroya, Efremenko, and Ta-Shma [2] give subexponential length locally decodable codes that can do better than random guessing for binary codes for δ fraction of corruption up to $\delta = 1/2-\alpha$ for any $\alpha > 0$, but the number of queries needed gets larger as δ gets closer to $1/2$.

1.1 Three query codes

Our main results show that achieving slightly larger than $1 - 3\delta$ correctness for 3-query locally decodable codes requires exponential length. We prove this for arbitrary (possibly nonlinear) binary codes and for linear codes over arbitrary finite fields. Note that larger, e.g. $1 - 2\delta$ correctness can be achieved even by 2-query linear codes: the Hadamard code is an example. With significantly larger number of queries, the correctness can be much higher as a function of δ (of the form $1 - \delta^{\Omega(q)}$): again the Hadamard code is an example. But this comes at the cost of having large length in the known constructions. Our results show that for 3-query codes, this increase in length cannot be avoided.

Here we give a somewhat simplified statement of the result for binary codes, without specifying the precise constants.

▶ **Theorem 1.2.** *Let* $\mathbf{C}$*:* $\{0,1\}^n \to \{0,1\}^m$ *be an arbitrary (possibly nonlinear) binary* $(3,\delta,\epsilon)$*-LDC with a nonadaptive decoder, and* n *large enough. If* $\frac{1}{2}+\epsilon > 1-3\delta+6\delta^2-4\delta^3+\phi(n)+\mu$*, where* $\phi(n) = O(1/n^{1/9})$*, then* $m \geq 2^{\Omega(\mu n^{1/3})}$*.*

We state the precise values hidden in the notation later in Theorem 3.1. We wanted to start with a more compact statement of our bound, showing that as soon as the correctness achieved by the code is above a certain threshold, the length of the codewords must be exponential. For binary codes this threshold is around $1 - 3\delta + 6\delta^2 - 4\delta^3$, which is just slightly larger than $1 - 3\delta$ for small values of δ. The value $1 - 3\delta$ is interesting, since there are subexponential length constructions of 3-query linear LDCs that achieve correctness $1-3\delta$ [7] and 3-query binary linear LDCs that achieve correctness slightly below $1-3\delta$ [19]. The value $1 - 3\delta + 6\delta^2 - 4\delta^3$ corresponds to the probability that the number of corrupted positions in a given triple is even, where the probability is over the distribution that corrupts each bit independently with probability δ.

For linear codes over arbitrary finite fields, we obtain stronger lower bounds. In our results for nonbinary codes, the value of the threshold is close to the threshold for the binary case, but slightly depends on the field size. We obtain exponential lower bounds for arbitrary finite fields, even if the field size depends on n.

We note that our bound holds for any $\delta \geq 0$ and any $0 < \epsilon \leq 1 - 1/|F|$, where δ and ϵ may be $o(1)$. For the bound to be nontrivial, we need $\delta \leq 1-1/|F|$, because of Observation

2.1. For n to be large enough for our purposes, it is sufficient if $\delta > \Omega(1/n^{1/9})$ and $\epsilon > \Omega(\frac{1}{n})$.

1.2 Arbitrary number of queries

We obtain similar results for arbitrary number of queries, under some assumptions on the decoding algorithm. We note that the types of decoding algorithms we consider have been commonly used in recent constructions. Our results explain the limitations on correctness of these constructions.

Unless otherwise noted, a q-query decoder is allowed to use less than q queries. So the correctness thresholds for requiring exponential length for q-query codes are never going to be smaller than the correctness thresholds for the same class of 3-query codes. In the special cases below, we show that the same thresholds to require exponential length as for 3-query codes also apply for arbitrary number of queries.

It remains open what is the correctness threshold (as a function of δ) to require exponential length for general q-query codes. Note that it will have to be a value larger than the threshold in our 3-query results: a q-query code for $q > 3$ can always do at least as well, as a 3-query code. We will see below, that if we require the query sets to be exactly of size q, then this is not necessarily the case.

1.2.1 Linear Decoders

One of the starting points of our approach was the observation that using larger number of queries does not help to tolerate errors if the decoder returns a fixed linear combination of the positions read. Moreover, the probability of error increases with the number of positions used with nonzero coefficients by a linear decoder. We formalize these ideas in our results about linear decoders.

▶ **Definition 1.3.** Let $\mathbf{C}$: $F^n \to F^m$ be an arbitrary (possibly nonlinear) code. We say that an algorithm A is a *linear decoder* for $\mathbf{C}$ if for any fixing of the outcomes of the coin flips of A, the value it returns is a fixed linear combination of the codeword positions it reads.

We show that linear decoders that use exactly q positions cannot achieve larger correctness than $1 - q\delta + o(\delta) + O(1/n)$, *regardless of the length of the code.* Moreover, we show that the correctness of any linear decoder, for any number of positions used, is at most $1-2\delta+o(\delta)+O(1/n)$. This holds for arbitrary (possibly nonlinear) codes and over any finite field F. This implies that our exponential length lower bounds extend to linear decoders with arbitrary number of queries, with the same correctness threshold as for 3-query codes.

Linear decoders are commonly used in the known constructions of locally decodable codes. In fact it is noted for example in [10, 18] that any (possibly nonlinear) binary (q, δ, ϵ)-LDC has a linear decoder that achieves correctness $1/2 + \epsilon/2^q$.

In the case of linear *smooth codes* (see [9]), requiring the decoders to be linear is inconsequential: for linear codes, if any algorithm gives nontrivial advantage over random guessing when querying a given set of codeword positions Q, then by Lemma 2.2, $e_i \in \text{span}(Q)$ must hold. Thus, there is a fixed linear combination of the positions in Q that gives the correct value of x_i for any input x. Using the same procedure as the original decoder to choose which positions to query and then returning this fixed linear combination (if it exists) cannot violate the smoothness of the code.

However, for locally decodable codes (both linear and nonlinear), requiring to use only linear decoders may significantly reduce the correctness associated with the code. For example, taking majorities, one can obtain correctness of the from $1 - \delta^{\Omega(q)}$. Our results show that in the recent results of [5, 2] obtaining subexponential length constructions with larger than $1-q\delta$ correctness for larger values of q, the use of nonlinear operations in the decoding algorithm is important.

Our results on linear decoders imply that there is no significantly better general reduction from smooth codes to locally decodable codes than the current bounds giving at most $1-q\delta$ correctness for q query locally decodable codes. The possibility of better reductions was raised in [9].

1.2.2 Matching sum decoders

Matching sum decoders are a subclass of linear decoders, thus our results on linear decoders immediately apply. However, for matching sum decoders we can prove stronger results. In particular, we can replace the correctness bounds $1 - q\delta + o(\delta) + O(1/n)$, by simply $1 - q\delta + O(1/n)$ for codes with q-query matching sum decoders regardless of the length of the code, and prove exponential lower bounds on the length of LDCs with matching sum decoders using any number of queries that achieve correctness larger than $1 - 3\delta + O(1/n)$.

Matching sum decoders were formally defined by Woodruff [19]. A q-query matching sum decoder picks a set of size q uniformly at random from a collection of sets that form a matching in the complete q-uniform hypergraph, whose vertices correspond to the positions of the codeword. Then, the decoder reads the positions corresponding to the chosen set, and returns the sum of the positions read. Most known constructions of locally decodable codes have such decoders.

Woodruff [19] proved that LDCs with 2-query matching sum decoders must have exponential length. We show that q-query matching sum decoders cannot achieve larger correctness than $1-q\delta+O(1/n)$, *regardless of the length of the code.* This holds for arbitrary codes and over any field.

Considering matching sum decoders where the query size is not fixed, we show that for any binary code (possibly nonlinear), and for linear codes over arbitrary finite fields, if a matching sum decoder with query sets of size at most q achieves correctness more than $1 - 3\delta + O(1/n^{1/3})$, then the length of the code must be exponential.

1.2.3 Query sets with large rank

For linear codes our proofs also apply to arbitrary number of queries, and possibly nonlinear decoders as long as the vectors corresponding to the positions queried are linearly independent. This is a property that holds in some of the known constructions of linear locally decodable codes. For such query sets, we show that if the correct value of x_i is spanned by q of the linearly independent vectors with nonzero coefficients, then the correctness of the decoder cannot be larger than $1 - q\delta + o(\delta) + O(1/n)$, *regardless of the length of the code.*

This implies, that for linear codes over arbitrary finite fields, if a q-query decoder (with query sets of size at most q) queries only linearly independent positions of the code and achieves correctness more than $1 - 3\delta + o(\delta) + O(1/n^{1/3})$, then the length of the code must be exponential. The exponential length lower bound extends to query sets that are not fully independent, but have large rank, with a correctness threshold that depends on the rank of the query sets. The results described for query sets with large rank are direct consequences of our proofs for linear codes.

1.3 Error Correcting Data Structures

Error correcting data structures were defined by de Wolf [17]. Such data structures are a variation of the traditional bit-probe model (see e.g. [11]), where the algorithms answering questions about the data are correct with probability at least $1/2 + \epsilon$, as long as at most δ fraction of the database representing the data is corrupted, possibly by adversarial error. It is noted in [17] that error correcting data structures for the membership problem yield locally decodable codes, with the same parameters. [17] showed the existence of error correcting data structures for the membership problem and some of its variants, assuming the existence of locally decodable codes with given parameters. Because of the direct correspondence between the two models, our results rule out the existence of error correcting data structures for membership of subexponential size with larger correctness than our thresholds above, for 3-probe algorithms, as well as for algorithms with arbitrary number of probes, assuming the algorithm only uses linear operations.

1.4 Techniques

We start by noting why some hand waiving arguments and intuition based on smooth codes would fail to explain our most general results. *Smooth codes* were defined by Katz and Trevisan [9], who also gave reductions between smooth codes and locally decodable codes. So up to changes in parameters, smooth codes and locally decodable codes are equivalent. Most of the current lower bounds for locally decodable codes have been proved via proving lower bounds for smooth codes, and the correctness of the known subexponential length constructions of 3-query linear LDCs is analyzed based on their property of having *smooth decoders*, that are correct with large probability if there is no error, and query each position of the code with not too large probability. However, the current techniques to analyze smooth decoders cannot imply larger than $1 - q\delta$ correctness for q-query locally decodable codes. In fact we show that no significantly better general reduction is possible. If we consider larger than $1 - q\delta$ correctness, then the equivalence of smooth codes and locally decodable codes starts to break down.

We elaborate on a few specific points below. One could try to argue that the probability that the decoder does not query any corrupted positions is upper bounded by a function not much larger than $1 - q\delta$, thus the decoder will have to read corrupted positions. However, errors may cancel out, so the fact that some of the positions read by the decoder may contain an error, in itself does not explain our lower bounds.

If the decoder was only working with query sets that form a matching, and the decoder was linear (which is the case in several of the known constructions), then - as we show - $1 - q\delta$ would in fact be a limit on correctness for decoders that query exactly q positions. But these assumptions do not have to hold for every decoding algorithm, and our results cannot be explained by this simplified view.

Our proofs of the 3-query lower bounds are based on a lemma that was central in obtaining the exponential lower bounds for 2-query codes. However, we would like to emphasize that we do not use 2-query lower bounds as a black box. We show that query sets that provide large correctness must contain subsets of size at most 2 that give nontrivial correlation with the input position we try to recover. But this does not imply that the code somehow “reduces” to a 2-query code. Consider the following simple example (many other examples are possible): query 3 positions such that each in itself has large correlation with the position x_i, and take the majority of the answers. Replacing this with reading only a subset of the bits, would preserve the properties of a smooth decoder, but it would reduce the correctness of the decoding algorithm. Thus, the decoder cannot be simply replaced by a 2-query decoder, if we want to preserve the correctness probabilities of the decoder.

Our approach can be summarized as follows: we show that in the case of 3-query codes, if the code is small, we can “force” the decoder to only examine query sets that are vulnerable to error. We achieve this by considering the algorithm's performance over random input x and a specially constructed distribution for the corruption caused by the adversary. We show that over our distribution, the decoder cannot perform much better than a linear decoder.

In all of our results, the probability of error is estimated in terms of the probability - over appropriate random corruption - of the event that the sum of the corruption in the positions of a given query set is nonzero. Intuitively, this probability would indeed give a lower bound on the error if the decoder always returned the sum (or a fixed linear combination) of the positions read, and if this was equal to the correct answer for uncorrupted codewords. For example, this would be the case for linear decoders of a linear code. However, we also consider nonlinear codes, and arbitrary decoders that may involve nonlinear operations. In fact, we do not claim that the probability of having nonzero sum of corruption in the query set is a lower bound on the error in general. Instead, we lower bound the probability of error by a different expression, and show that this expression is lower bounded by the probability mentioned above in the case of random corruption according to our distribution.

A crucial point in our proofs for nonlinear decoders (for both linear and nonlinear codes) is comparing the conditional probabilities of error of the decoder, conditioned on the sum of the values in the corrupted positions. We show that - under appropriate assumptions on the query sets for linear codes - the sum of these conditional probabilities of the decoder being incorrect, is always $|F| - 1$. A subtle point of this argument is that the various events we work with are not always independent. Our proof for nonlinear codes is based on a similar property of conditioning on the number of corrupted positions being odd vs. even. However, for nonlinear codes instead of directly considering the conditional probabilities of incorrect decoding, we reduce estimating the probability of error to estimating the probability that the sum of the positions read gives an incorrect answer. This analysis lets us estimate the probability of incorrect decoding even if the decoding algorithm uses nonlinear operations.

2 Preliminaries

The definition of locally decodable codes allows the decoding algorithm to be adaptive. Lower bounds for nonadaptive decoders can be translated to lower bounds for arbitrary decoders with the same number of queries but larger correctness: it is noted in the paper by Katz and Trevisan [9] that any adaptive (q, δ, ϵ) decoding algorithm for a code $\mathbf{C}\colon \Sigma^n \to \Gamma^m$, can be transformed to a nonadaptive $(q, \delta, \epsilon/|\Gamma|^{q-1})$ decoding algorithm for the same code.

We only consider nonadaptive decoding algorithms in the rest of the paper. We will refer to the (at most q) positions the algorithm chooses to read in a given invocation as a *query set*. In a nonadaptive algorithm, the choice of the query set only depends on the coin flips of the algorithm.

The following simple observation means that for proving lower bounds we may assume that $\delta < 1 - \frac{1}{|\Sigma|}$, since otherwise, no algorithm can do better than random guessing for *any* of the input positions.

▶ **Observation 2.1.** *Let A be a decoding algorithm for any code $\mathbf{C}\colon \Sigma^n \to \Sigma^m$. If $\delta \geq 1 - \frac{1}{|\Sigma|}$, then for any $i \in [n]$, $\min_{x\in\Sigma^n} \left(\min_{y\in\Gamma^m\,:\,d(y,\mathbf{C}(x))\leq\delta m} \Pr\left[A^y(i) = x_i\right]\right) \leq \frac{1}{|\Sigma|}$.*

For a linear code $\mathbf{C}\colon F^n \to F^m$, it is convenient to represent the function that determines a given codeword position by a vector: for $j \in [m]$, define $a_j \in F^n$ as the vector satisfying

$\forall x \in F^n$, $\mathbf{C}_j(x) = a_j \cdot x$. For vectors $a, x \in F^n$, we use $a \cdot x$ to denote their inner product over F. (We omit F from the notation.)

For a query set $Q = \{j_1, \ldots, j_q\} \subset [m]$, we use the notation $\mathrm{span}(Q)$ to represent the linear span of the vectors $a_{j_1}, \ldots, a_{j_q}$ corresponding to the positions in Q. We denote the i'th unit vector with length n by e_i. e_i has 1 in its i-th coordinate and 0 everywhere else.

The following lemma was stated in [8] for two query binary linear codes. Its extension to arbitrary fields and any number of queries is straightforward, but important for our arguments.

▶ **Lemma 2.2.** *(implicit in [8]) Let $\mathbf{C}$: $F^n \to F^m$ be a linear code. Let $i \in [n]$ and let $Q = \{j_1, j_2, ...j_q\} \subset [m]$ be a query set that the algorithm A queries with nonzero probability when trying to recover the value of input position i. Suppose $\Pr_{x \epsilon_U F^n}\left[A^{\mathbf{C}(x)}(i) = x_i \mid A \text{ queries } Q\right] > \frac{1}{|F|}$ where the probability is taken over letting x be uniformly random from F^n and over the internal coin tosses of A. Then $e_i \in \mathrm{span}(Q)$ must hold.*

We will use the following simple fact as well as Lemma 2.2 throughout our proof for linear codes.

▶ **Fact 2.3.** *(implicit in [1]) Let $a_1, \ldots, a_t$ be vectors from F^n. For x uniformly random from F^n, the corresponding random values $a_1 \cdot x, \ldots, a_t \cdot x$ are t independent uniformly distributed values from F, if and only if the vectors $a_1, \ldots, a_t$ are linearly independent over F.*

The following theorem of Goldreich, Karloff, Schulman and Trevisan [8] is a crucial ingredient of our proofs.

▶ **Theorem 2.4.** *[8] Let a_1, ... a_m be a sequence of (not necessarily distinct) elements of $\{0,1\}^n$ such that for every $i \in [n]$ there is a set M_i of disjoint pairs of indices $\{j_1, j_2\}$ such that $e_i = a_{j_1} \oplus a_{j_2}$. Then $m \geq 2^{2\alpha n}$, where $\alpha \triangleq \frac{\sum_{i=1}^n |M_i|}{nm}$.*

This theorem was extended to arbitrary finite fields in [8]. The dependence on the field size in the bound was removed by Dvir and Shpilka in [6]. We will use the following version (see Corollary 2.9 in [6]).

▶ **Theorem 2.5.** *[6] Let F be a field. Let a_1, ... a_m be a sequence of (not necessarily distinct) elements of F^n such that for every $i \in [n]$ there is a set M_i of disjoint pairs of indices $\{j_1, j_2\}$ such that $e_i \in \mathrm{span}(a_{j_1}, a_{j_2})$. Then $m \geq 2^{\alpha n - 1}$, where $\alpha \triangleq \frac{\sum_{i=1}^n |M_i|}{nm}$.*

A version of the theorem applicable to binary nonlinear codes is given in the "non-quantum" proof of the exponential lower bounds for 2-query binary nonlinear codes by Ben-Aroya, Regev and de Wolf [3].

▶ **Theorem 2.6.** *(implicit in Theorem 11 of [3]) Let $0 < \epsilon, \alpha < 1/2$. Let a_1, ... a_m be a sequence of (not necessarily distinct) functions from $\{0,1\}^n$ to $\{0,1\}$ such that for at least τn indices $i \in [n]$ there is a set M_i of disjoint pairs of indices $\{j_1, j_2\}$ such that $|M_i| \geq \alpha m$ and*

$$|\Pr_x[x_i = a_{j_1}(x) \oplus a_{j_2}(x)] - \Pr_x[x_i \neq a_{j_1}(x) \oplus a_{j_2}(x)]| \geq \epsilon$$

where the probability is over uniform $x \in \{0,1\}^n$. Then $m \geq 2^{\tau \alpha^2 \epsilon^2 n}$.

We also use the following theorem of Katz and Trevisan [9].

▶ **Theorem 2.7.** *(Theorem 2 in [9]) Let $\mathbf{C} : \{0,1\}^n \to R$ be a function. Assume there is an algorithm A such that for every $i \in [n]$, we have $\Pr_x[A(\mathbf{C}(x), i) = x_i] \geq \frac{1}{2} + \epsilon$, where the probability is taken over the internal coin tosses of A and uniform $x \in \{0,1\}^n$. Then $\log |R| \geq (1 - H(1/2 + \epsilon))n$.*

2.1 Notation

Let F be an arbitrary finite field. We denote by F^* the set of nonzero elements of F. Arithmetic operations involving field elements are over F. This should be clear from the context, and will be omitted from the notation.

For a code $\mathbf{C}\colon F^n \to F^m$, we can represent any vector $y \in F^m$ with $d(y, \mathbf{C}(x)) \leq \delta m$ as a sum of the form $y = \mathbf{C}(x) + B$, where $B \in F^m$, such that the number of nonzero entries in B is at most δm.

We will use the notation $\Pr_{x,B,A}$ to indicate probabilities over uniformly random input x from F^n, B chosen at random from a given distribution for corruption, and the random coin tosses of the given algorithm A.

Note that while in general the corruption may be produced by an arbitrary adversary, we will only consider distributions for B that do not depend on the input x or on the distribution for the coin tosses of the algorithm. This is sufficient for our purposes, since we are proving lower bounds on the length of the code.

3 Lower Bounds for Three Query Codes

3.1 Lower Bounds for Three Query Binary Codes

We state the precise version of our lower bound for arbitrary (possibly nonlinear) binary codes.

▶ **Theorem 3.1.** *Let $\mathbf{C}\colon \{0,1\}^n \to \{0,1\}^m$ be a $(3,\delta,\epsilon)$-LDC with a nonadaptive decoder, and n large enough. Let $\alpha \triangleq \delta - (\frac{1}{2} - (\frac{\epsilon}{4})^{1/3}) - (3/n^{1/3} + \frac{36}{n})^{1/3} - \nu$, and $\nu \triangleq \frac{10}{n(1-H(1/2+1/n^{1/3}))} = O(1/n^{1/3})$. If $\alpha > 0$, then $m \geq 2^{0.225\alpha^2 n^{1/3}}$.*

▶ Remark. We will show that $\alpha > 0$ when $\frac{1}{2} + \epsilon > 1 - 3\delta + 6\delta^2 - 4\delta^3 + \phi(n)$, where $\phi(n) = 4((3/n^{1/3} + \frac{36}{n})^{1/3} + \nu)$. Moreover, $\alpha > \frac{\mu}{4}$ when $\frac{1}{2} + \epsilon > \mu + 1 - 3\delta + 6\delta^2 - 4\delta^3 + \phi(n)$ for some $\mu \geq 0$. This implies the version of the bound stated in Theorem 1.2 for binary codes. Note that we could also obtain a lower bound of the form $2^{\Omega(n)}$ by setting ϵ_2 in the proof to a constant, but then the correctness required for the bound would be larger, roughly by $4(\epsilon_2)^{1/3}$.

Sketch of proof.

For the case of binary (possibly nonlinear) codes, using the Fourier representation of Boolean functions, and properties of correlation, we show that for any decoding algorithm, and for any query set Q, the advantage of the algorithm over random guessing when reading the values of the query set Q is at most the sum of the advantages obtained by all possible fixed linear functions over the given query set. This observation has been implicitly used also in the arguments of [10] and [20] showing the existence of linear decoders with correctness $1/2 + \epsilon/2^q$ for any binary (q,δ,ϵ)-LDC.

In all our proofs, we use a distribution for the adversary that corrupts each codeword position in a particular set S independently, and chooses the corruption over the remaining set of positions so that the total fraction of corrupted positions is still below δ. We construct the set S so that for query sets that do not intersect the set S, the contribution of sums over subsets of size at most 2 towards the advantage over random guessing is small. However, the size of S has to be small to keep the total fraction of corrupted positions below δ.

To achieve this, we first argue that in any LDC, the number of codeword positions that have large correlation with a given input bit x_i over random input x must be small for most input positions $i \in [n]$. This is straightforward for linear codes. Note however that for nonlinear codes, it is possible that a given codeword position has significant correlation with more than one input bit. We show the desired statement using Theorem 2.7.

Next we consider pairs of codeword positions, such that the sum of their values gives large correlation with a given input bit x_i over random input x. Using Theorem 2.6, we show that if the length of the code is small, then for most $i \in [n]$, all such pairs of positions can be covered by a small number of codeword positions.

This allows us to conclude that *if the length of the code is small*, then for at least one index $i \in [n]$, there exist a set S of small size, such that for query sets that do not intersect the set S, the contribution of sums over subsets of size at most 2 towards the advantage over random guessing the bit x_i is small.

We show that for any LDC with correctness $1/2 + \epsilon$, there is a decoding algorithm that never reads any of the positions in S, but is correct with probability at least $1/2 + \epsilon$ on average over random input x, and the random corruption of the above distribution. Note that the algorithm may not achieve the required correctness on every input and for every string within distance δm of $\mathbf{C}(x)$. We only claim a bound on its probability of being correct over uniformly random x and over random corruption according to our distribution.

This way we can argue that *if the length of the code is small* then there is a decoding algorithm that only uses query sets that either provide only small advantage over random guessing, or they involve 3 codeword positions, such that the sum of the 3 positions gives the correct value of the input bit x_i with large probability over random input and the random corruption according to our distribution.

On the other hand, for this decoding algorithm we can lower bound the probability of error by the probability that the sum of a given triple of codeword positions gives an incorrect value over random input and the random corruption according to our distribution.

Please see the full version of the paper for a detailed proof of the theorem and for the formal description of the distribution for the random corruption.

3.2 Lower Bounds for Three Query Linear Codes over Arbitrary Finite Fields

For linear codes over arbitrary finite fields, we obtain stronger lower bounds than our bounds for nonlinear codes.

Let F be an arbitrary finite field. We denote by F^* the set of nonzero elements of F. It is convenient to state the threshold on correctness in our bounds in terms of the probability of the event that a fixed linear combination of a given triple of coordinates of an appropriate random corruption equals to 0. More precisely, let $Q \subseteq [m]$ with $|Q| = q$ be an arbitrary fixed subset of the coordinates. Let $c_j \in F^*$, for $j \in Q$ and let $\delta \leq 1 - 1/|F|$. For the distributions we work with, the values of c_j will not make a difference, as long as they are all nonzero. Let $P(\delta, q, F) \triangleq \Pr_B \left[(\sum_{j \in Q} c_j B_j = 0)\right]$, where the probability is over $B \in F^m$ randomly chosen according to a distribution that first chooses to corrupt each coordinate in $[m]$ independently with probability δ, and then uniformly and independently assigns a value from F^* to each chosen coordinate of B. The remaining coordinates of B are set to 0. Note that an adversary using this distribution would possibly corrupt more than δm positions with nonzero probability, so this is not the distribution we use in our proofs. But it is convenient to use the probability $P(\delta, q, F)$ in the statement of our bounds, since $P(\delta, q, F)$ only depends on δ, q and $|F|$, it does not depend on m. The theorem also holds using the distribution that chooses δm positions uniformly (instead of independently corrupting the positions). While it is well known that these probabilities are not too far from each other in the two distributions, for our purposes we need more precise estimates.

We start with a simplified statement of the result without specifying the precise constants.

▶ **Theorem 3.2.** *Let* $\mathbf{C}\colon F^n \to F^m$ *be a linear* $(3, \delta, \epsilon)$*-LDC with a nonadaptive decoder, and* n *large enough. If* $\frac{1}{|F|} + \epsilon > P(\delta, 3, F) + \phi(n) + \mu$*, where* $\phi(n) = O(1/n^{1/3})$*, then* $m \geq 2^{\Omega(\mu n)}$*.*

We state the precise values hidden in the notation in Theorem 3.3.

For binary linear codes we present a slightly stronger bound in the next section.

▶ **Theorem 3.3.** *Let* $\mathbf{C}\colon F^n \to F^m$ *be a linear* $(q = 3, \delta, \epsilon)$*-LDC with a nonadaptive decoder,* $\delta \leq 1 - \frac{1}{|F|}$*, and* n *large enough. Then,* $m \geq 2^{.45\alpha n - 1}$ *where* $\alpha \triangleq \delta - (1 - \frac{1}{|F|} - \epsilon^{1/3}(1 - \frac{1}{|F|})^{2/3}) - (\frac{108}{n})^{1/3} - \frac{10}{n}$*.*

▶ Remark. We will show that $\alpha > 0$ when $\frac{1}{|F|} + \epsilon > 1 - 3\delta(1-\delta)^2 - (1 - \frac{1}{|F|-1})3\delta^2(1-\delta) - (1 - \frac{1}{|F|-1} + \frac{1}{(|F|-1)^2})\delta^3 + \phi(n)$, where $\phi(n) = 4((108/n)^{1/3} + 10/n)$. Moreover, $\alpha > \frac{\mu}{4}$ when $\frac{1}{|F|} + \epsilon > \mu + 1 - 3\delta(1-\delta)^2 - (1 - \frac{1}{|F|-1})3\delta^2(1-\delta) - (1 - \frac{1}{|F|-1} + \frac{1}{(|F|-1)^2})\delta^3 + \phi(n)$ for some $\mu \geq 0$. This implies the version of the bound stated in Theorem 3.2.

Sketch of proof.

Similarly to the proof for binary codes, we construct the set S of positions that we corrupt independently, so that for query sets that do not intersect the set S, the contribution of sums (or linear combinations) over subsets of size at most 2 towards the advantage over random guessing is small. Linear codes have the very strong property that linear combinations of codeword positions are either exactly equal to a given input bit, or give no advantage over random guessing towards recovering the given input bit (see Lemma 2.2). Thus, in the case of linear codes, we can construct the distribution of the adversary so that the decoding algorithm is left with query sets of size 3, such that no subsets of size at most 2 can give any advantage over random guessing. All other query sets that the algorithm can read will give no advantage over random guessing for a given input bit. Thus we don't need the part of the argument using Fourier representation of Boolean functions used in the binary proof to reduce estimating the probability of error to estimating the error over query sets of a special form. However, we still need to deal with the fact that the decoders can use nonlinear operations. We achieve this by considering the conditional probabilities of error of the decoder, conditioned on the sum (more precisely a fixed linear combination) of the values in the corrupted positions. In addition, we show that in the query sets we are left with the positions must correspond to linearly independent vectors in the generator matrix of the code. Based on this, we show that the probability of error (on average using our distribution) is lower bounded by $|F| - 1$ times the minimum over $k \in F$ of the probability that a fixed linear combination (with nonzero coefficients) of the corruption in the positions of the query set equals k.

See the full version for a detailed proof of the Theorem.

3.3 Lower Bounds for Three Query Binary Linear Codes

For binary linear codes, we obtain a slightly stronger bound than what follows from the lower bound for linear codes over arbitrary finite fields.

▶ **Theorem 3.4.** *Let* $\mathbf{C}\colon \{0,1\}^n \to \{0,1\}^m$ *be a linear* $(3, \delta, \epsilon)$*-LDC with a nonadaptive decoder, and* n *large enough. Then,* $m \geq 2^{1.8\alpha n}$ *where* $\alpha \triangleq \delta - (\frac{1}{2} - (\frac{\epsilon}{4})^{1/3}) - (\frac{36}{n})^{1/3} - \frac{10}{n}$*.*

▶ Remark. We will show that $\alpha > 0$ when $\frac{1}{2} + \epsilon > 1 - 3\delta + 6\delta^2 - 4\delta^3 + \phi(n)$, where $\phi(n) = 4((36/n)^{1/3} + 10/n)$. Moreover, $\alpha > \frac{\mu}{4}$ when $\frac{1}{2} + \epsilon > \mu + 1 - 3\delta + 6\delta^2 - 4\delta^3 + \phi(n)$ for some $\mu \geq 0$. This implies the version of the bound stated in Theorem 3.2 for binary codes.

The proof is almost identical to the proof in the previous section for arbitrary finite fields. The improvement comes from using Theorem 2.4, and because in the case of binary linear codes we can use a node cover of size $|M_1|$ instead of $2|M_1|$ when defining the distribution.

Acknowledgements We thank David Woodruff for helpful conversations and for pointing us to [3], and the anonymous referees for helpful comments.

References

1 N. Alon, L. Babai and A. Itai: A fast and simple randomized parallel algorithm for the maximal independent set problem. *Journal of Algorithms*, Vol. 7, pp. 567 - 583, 1986.

2 A. Ben-Aroya, K. Efremenko, A. Ta-Shma: Local list-decoding with a constant number of queries. ECCC Report No. 47, 2010.

3 A. Ben-Aroya, O. Regev and R. de Wolf: A hypercontractive inequality for matrix valued functions with applications to quantum computing and LDCs. In *Proceedings of FOCS 2008*, pp. 477 - 486.

4 Z. Dvir: On matrix rigidity and locally self-correctable codes. ECCC Technical Report No. 134, 2009.

5 Z. Dvir, P. Gopalan, S. Yekhanin: Matching Vector Codes. In *Proceedings of FOCS 2010*, pp. 705 - 714.

6 Z. Dvir and A. Shpilka: Locally decodable codes with 2 queries and polynomial identity testing for depth 3 circuits. In *Proceedings of STOC 2005*, pp. 592 - 601.

7 K. Efremenko: 3-query locally decodable codes of subexponential length. In *Proceedings of STOC 2009*, pp. 39-44.

8 O. Goldreich, H. Karloff, L. Schulman and L. Trevisan: Lower bounds for linear locally decodable codes and private information retrieval. *Comput. Complex.*, 15(3):263–296, 2006.

9 J. Katz and L. Trevisan: On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of STOC 2000*, pp. 80-86.

10 I. Kerenidis and R. de Wolf: Exponential lower bound for 2-query locally decodable codes via a quantum argument. In *Proceedings of STOC 2003*, pp. 106-115.

11 P. Bro Miltersen: Cell probe complexity - a survey. In *Advances in Data Structures Workshop, 1999.*

12 K. Obata: Optimal lower bounds for 2-query locally decodable linear codes. In *Proceedings of RANDOM 2002* pp. 39–50.

13 P. Raghavendra: A note on Yekhanin's locally decodable codes. *ECCC TR07-016*, 2007.

14 D. Shiowattana and S. V. Lokam: An optimal lower bound for 2-query locally decodable linear codes. *Inf. Process. Lett.*, 97(6):244–250, 2006.

15 L. Trevisan: Some applications of coding theory in computational complexity. *ECCC TR04-043*, 2004.

16 S. Wehner and R. de Wolf: Improved lower bounds for locally decodable codes and private information retrieval. In *Proceedings of ICALP 2005* Vol. 3580 of LNCS, pp. 1424 - 1436.

17 R. de Wolf: Error-correcting data structures In *Proceedings of STACS 2009* pp. 313 - 324.

18 D. Woodruff: Some new lower bounds for general locally decodable codes. *ECCC TR07-006*, 2006.

19 D. Woodruff: Corruption and recovery-efficient locally decodable codes. In *Proceedings of RANDOM 2008*, pp. 584–595.

20 D. Woodruff: A Quadratic Lower Bound for Three-Query Linear Locally Decodable Codes over Any Field. In *Proceedings of RANDOM 2010.*

21 S. Yekhanin: Towards 3-query locally decodable codes of subexponential length. In *Proceedings of STOC 2007*, pp. 266–274.

Index of Authors

Ackermann, Marcel R. 308
Albers, Susanne 1
Antoniadis, Antonios 531
Aschinger, Markus 12

Babenko, Maxim 519
Bienvenu, Laurent 452
Blanchet-Sadri, Francine 225
Bläser, Markus 555
Blömer, Johannes 308
Bodlaender, Hans L. 165, 177
Bojańczyk, Mikołaj 105
Borello, Alex 273
Bušić, Ana 296

Caminiti, Saverio 45
Case, John 320
Chalopin, Jérémie 153
Chan, Sze-Hang 392
Childs, Andrew M. 661
Comon-Lundh, Hubert 29
Cortier, Véronique 29

Datta, Samir 579
Demaine, Erik D. 201
Dereniowski, Dariusz 416
Disser, Yann 153
Drescher, Conrad 12
Dumitrescu, Adrian 637
Dyer, Martin 261

Eggermont, Christian E.J. 484
Engels, Christian 555

Fatès, Nazim 284
Figueira, Diego 93
Finocchi, Irene 45
Fomin, Fedor V. 189
Freydenberger, Dominik D. 213, 507
Fusco, Emanuele G. 45

Gál, Anna 673
Ganian, Robert 404
Giakkoupis, George 57
Gottlob, Georg 12
Grenet, Bruno 543
Gulan, Stefan 495

Guo, Heng 249
Gusakov, Alexey 519

Halldórsson, Magnús M. 472
He, Xin 141
Hertli, Timon 237
Hliněný, Petr 404
Huang, Sangxia 249
Hüffner, Falk 531

Jansen, Bart M. P. 165, 177
Jeavons, Peter 12

Kallas, Jakub 356
Kaltofen, Erich L. 543
Kaplan, Haim 117
Karloff, Howard 332
Kelner, Jonathan A. 440
Knauer, Christian 649
Koiran, Pascal 543
Kolman, Petr 129
Kor, Liah 69
Korman, Amos 69
Korn, Flip 332
Kothari, Robin 661
Kötzing, Timo 320
Kratsch, Stefan 165
Kufleitner, Manfred 356
Kulkarni, Raghav 579
Kuntze, Daniel 308
Kuperberg, Denis 627
Kupferman, Orna 615

Lam, Tak-Wah 392
Lauser, Alexander 356
Lee, Lap-Kei 392
Lensmire, John 225
Lenzner, Pascal 531
Levin, Alex 440
Lokshtanov, Daniel 189
López-Ortiz, Alejandro 380
Lu, Pinyan 249
Lustig, Yoad 615

Mairesse, Jean 296
Makarychev, Konstantin 332
Marcovici, Irène 296

McGregor, Andrew 428
Merkle, Wolfgang 452
Mertzios, George B. 591
Mihalák, Matúš 153
Mills, Andrew 673
Misra, Neeldhara 189
Moldenhauer, Carsten 531
Moser, Robin A. 237
Mundhenk, Martin 368

Nevisi, Hossein 213
Nies, André 452
Nussbaum, Yahav 117

Obdržálek, Jan 404

Parys, Paweł 603
Patitz, Matthew J. 201
Patt-Shamir, Boaz 472
Peleg, David 69
Philip, Geevarghese 189
Portier, Natacha 543

Qiao, Youming 567
Quimper, Claude-Guy 380

Rabani, Yuval 332
Rawitz, Dror 472
Reidenbach, Daniel 213
Richard, Gaétan 273
Richerby, David 261
Rudra, Atri 428
Rumyantsev, Andrey Yu. 464

Sarma M.N., Jayalal 567
Saurabh, Saket 189
Scheder, Dominik 237
Scheideler, Christian 129
Schrijver, Alexander 484
Schulz, André 637
Schweller, Robert T. 201
Segoufin, Luc 81, 93, 344
Sheffer, Adam 637
Sohler, Christian 308
Souza, Alexander 531
Summers, Scott M. 201

Tang, Bangsheng 567
ten Cate, Balder 344
Terrier, Véronique 273
Tewari, Raghunath 579
Thorstensen, Evgenij 12
Tiwary, Hans Raj 649
Toruńczyk, Szymon 81
Tóth, Csaba D. 637

Uurtamo, Steve 428

Vardi, Moshe Y. 615
Vinodchandran, N. V. 579

Wang, Jiun-Jie 141
Weiß, Felix 368
Werner, Daniel 649
Widmayer, Peter 153
Woeginger, Gerhard J. 484

Xia, Mingji 249

Yannakakis, Mihalis 615