29th International Symposium on Theoretical Aspects of Computer Science

STACS'12, February 29th to March 3rd, 2012, Paris, France

Edited by

Christoph Dürr

Thomas Wilke

Editors

Christoph Dürr
CNRS and Laboratoire d'Informatique de Paris-6
Université Pierre et Marie Curie
christoph.durr@lip6.fr

Thomas Wilke
Institut für Informatik
Christian-Albrechts-Universität zu Kiel
wilke@ti.informatik.uni-kiel.de

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 978-3-939897-35-4

Published online and open access by
Schloss Dagstuhl – Leibniz-Center for Informatics GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at http://www.dagstuhl.de/dagpub/978-3-939897-35-4.

Publication date
February, 2012

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 .

Digital Object Identifier: 10.4230/LIPIcs.STACS.2012.i

ISBN 978-3-939897-35-4 **ISSN 1868-8969** **http://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

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

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 from February 29th to March 3rd, held in Paris, is the 29th 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), Nancy (2010) and Dortmund (2011).

The interest in STACS has remained at a high level over the past years. The STACS 2012 call for papers led to 273 submissions from 38 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.

This year the conference included a tutorial. We would like to express our thanks to the speaker R. Ravi for this tutorial, as well as as to the invited speakers, Thomas Colcombet, Martin Dietzfelbinger and Shafi Goldwasser. Special thanks go to Andrei Voronkov for his EasyChair software (www.easychair.org). Moreover, we would like to warmly thank Evripidis Bampis, Olivier Dubois, Noura El-Habchi and Thierry Lanfroy for continuous help throughout the conference organization.

For the 5th time, STACS proceedings are published in electronic form. Instead of providing a complete printed version, we allowed participants to print selected papers on place. The electronic proceedings are available through the LIPIcs (Leibniz International Proceedings in Informatics) and the HAL (hyper articles en ligne) series. Both, HAL and LIPIcs, guarantee perennial, free and easy electronic access, while the authors retain the rights over their work. STACS 2012 received funds from the lab LIP6 at the Université Pierre et Marie Curie and the GdR Informatique et Mathématique. We thank them for their support!

February 2012 Christoph Dürr and Thomas Wilke

29th International Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Conference Organization

Program Comittee

Vikraman Arvind	Chennai
Manuel Bodirsky	Ecole Polytechnique
Hans Bodlaender	Utrecht
Felix Brandt	Munich
Véronique Bruyère	Mons
Didier Caucal	Marne-la-Vallée
Stéphane Demri	Cachan
Christoph Dürr	Paris (co-chair)
Robert Elsässer	Paderborn
Anupam Gupta	Pittsburgh
Lane A. Hemaspaandra	Rochester
Stephan Kreutzer	Berlin
Orna Kupferman	Jerusalem
Dietrich Kuske	Ilmenau
Seffi Naor	Technion
Monaldo Mastrolilli	Manno-Lugano
Michel de Rougemont	Paris
Michiel Smid	Ottawa
Iain Stewart	Durham
Heribert Vollmer	Hannover
Igor Walukiewicz	Bordeaux
Thomas Wilke	Kiel (co-chair)

Local Organization Comittee

Evripidis Bampis
Olivier Dubois
Christoph Dürr
Noura El Habchi

29th International Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke

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

External Reviewers

Faisal Abukhzam
Luca Aceto
Susanne Albers
Cyril Allauzen
Peter Allen
Eric Allender
Jean-Paul Allouche
Shaull Almagor
Greg Aloupis
Helmut Alt
Andris Ambainis
Etienne Andre
Spyros Angelopoulos
Holger Arndt
Mohamed Faouzi Atig
Man Ho Au
Haris Aziz
Yoram Bachrach
Ashwinkumar Badanidiyuru
David Baelde
Evripidis Bampis
Nikhil Bansal
Vince Barany
Pablo Barceló
David Mix Barrington
Frederique Bassino
Mohammadhossein Bateni
Marie-Pierre Béal
Nicolas Bedon
Petra Berenbrink
Anna Bernasconi
Nathalie Bertrand
Alexis Bes
Olaf Beyersdorff
Arnab Bhattacharyya
Vittorio Bilò
Stephen Binns
Markus Bläser
Achim Blumensath
Carlo Blundo
Olivier Bodini
Elmar Böhler
Beate Bollig
Benedikt Bollig
Bernd Borchert
Glencora Borradaile
Olivier Bournez
Julian Bradfield
Laurent Braud
Thomas Brihaye
Markus Brill
Hajo Broersma
Niv Buchbinder
Harry Buhrman
Laurent Bulteau
Alan Bundy
Michael Butler
Jaroslaw Byrka
Leizhen Cai
Cristian S. Calude
Ioannis Caragiannis
Arnaud Carayol
Jean Cardinal
Olivier Carton
Julien Cassaigne
Daniele Catanzaro
Douglas Cenzer
Jeremie Chalopin
Timothy M. Chan
Sunil Chandran
Krishnendu Chatterjee
Hong-Bin Chen
Yijia Chen
Mahdi Cheraghchi
Christian Choffrut
Sung-Soon Choi
Eisentraut Christian
Asaf Cohen
Johanne Cohen
Amin Coja-Oghlan
Thomas Colcombet
Atlas Cook Iv
Colin Cooper
Andreas Cord-Landwehr
Nadia Creignou
Maxime Crochemore
Marek Cygan
Gianlorenzo d'Angelo
Deepak D'Souza
Juergen Dassow
Samir Datta
Hervé Daudé
Annalisa de Bonis
Bart de Keijzer
Alessandro de Luca
Ronald de Wolf
Holger Dell
Carole Delporte-Gallet
Josee Desharnais
Pietro di Lena
Martin Dietzfelbinger
Laurent Doyen
Feodor Dragan
Manfred Droste
Stefan Droste
Andrew Drucker
Clemens Dubslaff
Philippe Duchon
Marc Ducobu
Jacques Duparc
Arnaud Durand
Sagarmoy Dutta
Johannes Ebbing
Omer Egecioglu
Michael Elberfeld
Matthias Englert
David Eppstein
Leah Epstein
Aytek Erdil
Lawrence Erickson
Javier Esparza
Alessandro Facchini
Michael Fellows
Henning Fernau
Yuval Filmus
Irene Finocchi
Felix Fischer
Michele Flammini
Paola Flocchini
Jörg Flum
Fedor V. Fomin
Bernard Fortz
Dimitris Fotakis
Pierre Fraigniaud
Mathew Francis
Natalie Frank
Johanna Franklin
Tom Friedetzky
Emanuele Guido Fusco
Martin Gairing
Julia Gamzova

29th International Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Iftah Gamzu
Sumit Ganguly
Pierre Ganty
Naveen Garg
Pawel Gawrychowski
Viliam Geffert
Chryssis Georgiou
George Giakkoupis
Panos Giannopoulos
Hugo Gimbert
Christian Glasser
Stefan Göller
Petr Golovach
Valentin Goranko
Jean Goubault-Larrecq
Kalman Graffi
Gianluigi Greco
Martin Grohe
Jiong Guo
Greg Gutin
Gus Gutoski
Peter Habermehl
Michel Habib
Matthew Hague
Vesa Halava
Kristoffer Arnsfelt Hansen
Tero Harju
Paul Harrenstein
Aram Harrow
Prahladh Harsha
Niko Haubold
Sikander Hayat
Pinar Heggernes
Lauri Hella
Brett Hemenway
Ulrich Hertrampf
Volker Heun
Daniel Hirschkoff
Martin Hoefer
Thomas Hofmeister
Hendrik Jan Hoogeboom
Han Hoogeveen
Michael Huber
Falk Hüffner
Martina Hüllmann
Paul Hunter
Martin Huschenbett
Csanad Imreh
Malika Izabachene
Bart Jansen
Klaus Jansen
Maurice Jansen
Albert Xin Jiang
Barbara Jobstmann
Pushkar Joglekar
Daniel Johannsen
Jan Johannsen
Matthew Johnson
Gwenaël Joret
Vincent Jost
Tomasz Jurdzinski
Valentine Kabanets
Lukasz Kaiser
Marcin Kamiński
Daniel Kane
Mihyun Kang
Haim Kaplan
Juhani Karhumaki
Lila Kari
Alexander Kartzow
Ian Kash
Ioannis Kassios
Neeraj Kayal
Julia Kempe
Barbara Kempkes
Iordanis Kerenidis
Bakhadyr Khoussainov
Daniel Kirsten
Bruce Kitchens
Felix Klaedtke
Ralf Klasing
Hartmut Klauck
Kim-Manuel Klein
Ton Kloks
Teodor Knapik
Christian Knauer
Sebastian Kniesburges
Johannes Koebler
Matthias Koeppe
Ekkehard Köhler
Christian Konrad
Steffen Kopecki
Matias Korman
Nitish Korula
Miroslaw Korzeniowski
Sven Kosub
Timo Kötzing
Ioannis Koutis
Andreas Koutsopoulos
Daniel Kral
Rastislav Kralovic
Jan Kratochvil
Dieter Kratsch
Stefan Kratsch
Veronika Kraus
Andreas Krebs
Stephan Krenn
Ravishankar Krishnaswamy
Michael Krivelev
Sebastian Kuhnert
Raghav Kulkarni
Oliver Kullmann
Amit Kumar
Sergey Kuniavsky
Piyush Kurur
Martin Kutrib
Arnaud Labourel
Oded Lachish
Christiane Lammersen
Jérôme Lang
Sławomir Lasota
Ranko Lazic
Jean-Marie Le Bars
Francois Le Gall
Marion Le Gonidec
Thierry Lecroq
Guillaume Lecué
Lap-Kei Lee
Axel Legay
Hans Leiß
Steffen Lempp
Christoph Lenzen
Jerome Leroux
Asaf Levin
Shi Li
Nutan Limaye
Andrzej Lingas
Christof Loding
Christof Loeding
Maarten Löffler
Peter Lohmann
Markus Lohrey
Satyanarayana Lokam
Daniel Lokshtanov
Sylvain Lombardy
Alex Lopez-Ortiz
Martin Lotz
Ronny Luss
Jack H. Lutz
Florent Madelaine

Fréderic Magniez
Meena Mahajan
Janos Makowsky
Andreas Maletti
Bodo Manthey
Maurice Margenstern
Barnaby Martin
Conrado Martinez
Dániel Marx
Alexander May
Elvira Mayordomo
Richard Mayr
Arne Meier
Wolfgang Merkle
George Mertzios
Antoine Meyer
Tom Meyerovitch
Alexej Miasnikov
Daniele Micciancio
Dimitrios Michail
Christian Michaux
Peter Bro Miltersen
Matthias Mnich
Tobias Moemke
Ankur Moitra
Morteza Monemizadeh
Ashley Montanaro
Pat Morin
Christophe Morvan
Benjamin Moseley
Ahuva Mu'Alem
Partha Mukhopadhyay
Haiko Müller
Julian-Steffen Müller
Mike Müller
Martin Mundhenk
S. Muthukrishnan
Nikolaus Mutsanas
Viswanath Nagarajan
Shin-Ichi Nakano
Subhas Nandy
Nina Narodytska
François Nathanael
Ofer Neiman
Cyril Nicaud
Peter Niebert
Rolf Niedermeier
Prajakta Nimbhorkar
Tim Nonner
Gustav Nordh
Jan Obdrzalek
Adrian Ogierman
Nicolas Ollinger
Fukuhito Ooshita
Yota Otachi
Angelica Pachon
Renato Paes Leme
Konstantinos Panagiotou
Prakash Panangaden
Debmalya Panigrahi
Gyula Pap
Christophe Paul
Daniel Paulusma
Paolo Penna
Yury Person
David Phillips
Claudine Picaronny
Giovanni Pighizzini
Marcin Pilipczuk
Michael Pinsker
Maria Polukarov
Ely Porat
Stefan Porschen
Lars Prädel
Robert Preis
Gabriele Puppis
Artem Pyatkin
Evangelia Pyrga
Bruno Quoitin
Jaikumar Radhakrishnan
Harald Raecke
Rajeev Raman
Narad Rampersad
Jean-François Raskin
Dror Rawitz
Alexander Reich
Jan Reimann
Klaus Reinhardt
Rogério Reis
Steffen Reith
Eric Remila
Guenael Renault
Bernard Ries
Simone Rinaldi
Giuseppina Rindone
Michael Rink
Chloé Rispal
Jérémie Roland
Andrei Romashchenko
Adi Rosen
Mark Rudelson
Mathieu Sablik
Chandan Saha
Hiroshi Sakamoto
Michael Saks
Sylvain Salvati
Peter Sanders
Miklos Santha
Rahul Santhanam
Jayalal Sarma M.N.
Srinivasa Rao Satti
Thomas Sauerwald
Mathias Schacht
Michael Schapira
Sven Schewe
Lena Schlipf
Stefan Schmid
Thomas Schneider
Henning Schnoor
Ulrich Schöpp
Yevgeny Schreiber
Roy Schwartz
Pascal Schweitzer
Hans Georg Seedig
Danny Segev
Pavel Semukhin
Geraud Senizergues
Olivier Serre
Hadas Shachnai
Jeffrey Shallit
Bruce Shepherd
Arseny Shur
Florian Sikora
Pierre Simonnet
Rene Sitters
Primoz Skraba
Shakhar Smorodinsky
Christian Sohler
Roberto Solis-Oba
Shay Solomon
Alexander Souza
Daniel Spielman
Srikanth Srinivasan
B. Srivathsan
Juraj Stacho
Ludwig Staiger
Georgios Stamoulis
Alexandre Stauffer
Frank Stephan
Lorna Stewart

Lutz Strassburger
K. Subramani
C.R. Subramanian
Maxim Sviridenko
Guido Tack
Masayuki Takeda
Tami Tamir
Val Tannen
Till Tantau
Siamak Tazari
Stefano Tessaro
Raghunath Tewari
Johan Thapper
Thomas Thierauf
Dimitrios Thilikos
Rick Thomas
Ioan Todinca
Jacobo Torán
Leen Torenvliet
Szymon Toruńczyk
Denis Trystram
Hao-Kuan Tso
Dekel Tsur
Paolo Turrini
Sarvagya Upadhyay
René Van Bevern
Franck Van Breugel
Marc Van Kreveld
Rob Van Stee
Vinodchandran Variyam
Oscar Vasquez
Yadu Vasudev
Mikael Vejdemo-Johansson
Stéphane Vialette
Laurent Viennot
Nisheeth Vishnoi
Mahesh Viswanathan
Berthold Voecking
Jan Vondrak
Vladimir Vyugin
Dorothea Wagner
Klaus Wagner
Uli Wagner
Magnus Wahlstroem
Magnus Wahlström
Yajun Wang
John Watrous
Pascal Weil
Oren Weimann
Amit Weinstein
Udi Wieder
Virginia Williams
Carola Winzen
Steve Wismath
Maximilian Witek
Dominik Wojtczak
David Xiao
Jinbo Xu
Jinhui Xu
Jonathan Yaniv
Amir Yehudayoff
Hsu-Chun Yen
Ke Yi
Sheng Yu
Konrad Zdanowski
Norbert Zeh
Xavier Zeitoun
Mariano Zelke
Rico Zenklusen
Qin Zhang
Shengyu Zhang
Sandra Zilles
Jens Zumbraegel

Contents

Invited talks

Regular contributions

29th International Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Forms of Determinism for Automata*

Thomas Colcombet[1]

1 **Liafa / CNRS / Université Paris Diderot**
Case 7014, F-75205 Paris Cedex 13, France
thomas.colcombet@liafa.jussieu.fr

Abstract

We survey in this paper some variants of the notion of determinism, refining the spectrum between non-determinism and determinism. We present unambiguous automata, strongly unambiguous automata, prophetic automata, guidable automata, and history-deterministic automata. We instantiate these various notions for finite words, infinite words, finite trees, infinite trees, data languages, and cost functions. The main results are underlined and some open problems proposed.

1998 ACM Subject Classification F.1.1 Models of Computation, F.1.2 Modes of Computation, F.4.3 Formal Languages

Keywords and phrases Automata, determinism, unambiguity, words, infinite trees

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.1

1 Introduction

The relationship between deterministic and non-deterministic machines plays a central role for many questions in computer science, in particular, when it comes to complexity. In this survey, we investigate these notions and related ones in the context of classical automata and more recent automata models.

The role of determinism for automata is different from its role for general Turing machines. The reason for that is that automata are more considered as data structures than as programs. Indeed, automata are meant to be compact representations of (possibly infinite) languages. They are meant to be transformed, composed and used in *decision procedures*. General Turing machines cannot play such a role since by the theorem of Rice only trivial properties concerning them can be decided. For this essential reason, determinism or non-determinism are considered under a very different angle for automata compared to, say, complexity theory.

Since the seminal works of Myhill [23] and Rabin and Scott [26] deterministic and non-deterministic finite-state automata are known to define the same classes of languages over finite words. It is also known that they have very different properties. For instance, complementing a deterministic automaton is a straighforward operation (linear time and space), while it is exponential for general non-deterministic automata in the sense that there exist languages accepted by an automaton which when complemented requires an exponentially bigger automaton for being accepted. For this reason it is meaningful to carefully distinguish the two models as far as decision procedures are involved.

As automata theory developed over time, new types of automata were introduced such as automata over infinite words, finite trees, infinite trees, etc... The relationship between

* This work has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement 259454.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 1–23

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

deterministic and non-deterministic machines became richer at each step, reflecting the intricate mathematical principles underlying these objects.

In this survey, the following objects will be encountered:

- Standard word automata.
- Register automata (similar to the model of finite-memory automata of Kaminsky and Francez). Such automata read words over an infinite alphabet containing data values (such as natural numbers), can store such data values in a finite number of registers, and compare them for equality with the current data value.

A typical example is the language: 'some letter appears twice in the word'.

- Automata over infinite words (of length ω). Due to the infinite length of the words, such automata do not use accepting states. The accepting runs are defined in terms of the states that appear infinitely often. Prominent examples are Büchi and Müller automata.

A typical example is the language: 'the letter a appears finitely many times'.

- Automata over finite trees. Such automata can branch; thus computations proceed 'concurrently' in several subtrees of the tree.

A typical example the language is: 'every branch contains an occurrence of letter b'.

- Automata over infinite trees. As with finite trees such automata are branching, but additionaly the run should satisfy an accepting condition on each of its branches, similarly to the case of infinite words.

A typical example is the language: 'the letter a occurs at infinitely many nodes'.

- Cost automata. These automata are used for recognising functions from words to non-negative integers. Such automata use several counters that can be incremented and reset. The values taken by the counters during all runs are aggregated into a value which is output.

A typical example is the function: 'the number of occurrences of letter a'.

We present in this survey different notions of determinism and non-determinism. It is important to understand that these variants are not tied to a specific form of automata. The different models are used to exhibit the differences between the notions, and also to illustrate why each of these notions is interesting in practice. Thus, we will introduce several restrictions to non-deterministic automata which are not as restrictive as determinism, but still allow us to derive some useful properties. There are several motivations for considering such variants.

Complexity. A first reason for preferring deterministic automata is that some operations are easier to perform with deterministic automata. This is the case for universality testing (PTIME against PSPACE in general over words), or complement (linear blowup against exponential blowup over words). What kind of notions, less restrictive than determinism yields the same complexity results? The notions of unambiguity and strong unambiguity provide some answers.

Decidability. A second reason is that one works with a class of automata which does not admit determinisation (i.e., deterministic automata are strictly weaker than non-deterministic ones) and that the non-deterministic automata do not enjoy good properties. This is the case with register automata. Non-deterministic register automata have an undecidable universality problem while deterministic ones have a decidable universality problem. One is interested in this situation in introducing forms of determinism, not as restrictive as the general determinism, and for which the universality problem remains decidable. It is relevant in this case to consider the class of strongly unambiguous automata as a class of automata of intermediate expressive power which retain good algorithmic properties.

Structure. Another reason for preferring deterministic automata to non-deterministic automata is that they have a sharper structure for some advanced decision procedures. This

happens, for instance, when one is interested in the characterisation of some sub-classes of the regular languages, *i.e.* problems such as, 'is it possible to define a language given in input in some fixed fragment of logic?'. Deterministic automata, being more constrained, exhibit more behaviour in their structure, and for this reason are better indicated. We will see that prophetic automata over infinite words are good in this respect for characterising future temporal logics. In a similar way, guidable automata are suitable for analysing the fix-point structure of languages of infinite trees.

Games. A last reason is that one wants to use a specific property of deterministic automata, which is not enjoyed by non-deterministic automata. This is in particular the case for composing with games. The idea behind this is that in a game (say, two players, turn based), the first player is not aware of the future of the play since it depends of the choices of the opponent. Hence, he has to decide his moves solely based on the past of the play. The determinism of an automaton has a similar flavour since it means making decisions solely based on the current state, and hence without guessing any information concerning the future. This similarity can be used for safely composing games with automata. This is an important technique in automata theory. We will see that in this context the notion of history-determinism (also called 'good for solving games') can be used as a replacement to determinism. A motivating example is the theory of cost functions where deterministic automata are strictly weaker than general automata while history-deterministic ones are as expressive.

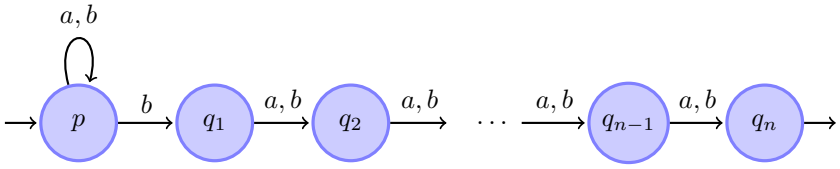
Through these motivations, many different variations around determinism have emerged: unambiguity, strong unambiguity, prophetic automata, guidability and history-determinism. Their presentation is organised as follows.

In Section 2 we recall the definitions of word automata as well as some standard results for them. In Section 3 we consider several notions of unambiguity. More specifically, we introduce unambiguity and strong unambiguity, and we study these notions over word automata, register automata, infinite word automata (in particular prophetic Büchi automata) and infinite tree automata (in particular the inherent ambiguity result). In Section 4 we present guidable automata, a notion related to top-down determinism over finite and infinite trees. In Section 5 we introduce history-deterministic automata, also described as those 'good for solving games'. We define the notion and give results over finite words. We then present the motivating example of cost functions. We also explain why these automata behave nicely in the context of games.

2 Word automata

A *(non-deterministic) (finite word) automaton* $(Q, \mathbb{A}, I, \Delta, F)$ consists of a finite set of *states* Q, an alphabet $\mathbb{A}$, a set of *initial states* $I \subseteq Q$, a *transition relation* $\Delta \subseteq Q \times \mathbb{A} \times Q$, and a set of *final states* $F \subseteq Q$. A *run* of the automaton over a word $a_1 \dots a_n$ is a sequence $q_0, \dots, q_n$ of states such that $(q_{i-1}, a_i, q_i) \in \Delta$ for all $i = 1 \dots n$. The run is *initial* if $q_0 \in I$ and it is final if $q_n \in F$. A run is *accepting* if it is both initial and final. A word is *accepted* if there is an accepting run of the automaton over it. The set of accepted words is called the *language* of the automaton. One also says that this is an automaton *for* the language. We will use the same terminology for extended forms of automata.

Consider for instance the (non-deterministic) automaton with input alphabet $\{a, b\}$ depicted as follows:



Following tradition, a transition (p, a, q) is denoted by an edge from p to q labeled a. Multiple transitions going between the same states are denoted with commas separating letters. Initial states have an incoming arrow without origin, and final states have a similar outgoing arrow. The illustrated automaton is for the language 'the nth letter from the end is a b'.

An automaton is called *deterministic* if for all states p and all letters a there is at most one transition of the form (p, a, q). Additionally it is called complete if for all states p and letter a there exists a transition of the form (p, a, q). The symmetric notion is co-determinism. An automaton is *co-deterministic* if for all states q and all letters a there is at most one transition of the form (p, a, q). The automaton above is co-deterministic, but not deterministic. The *size* of an automaton is number of its states.

▶ **Theorem 1.** *Word automata of size n can be transformed into deterministic and complete automata of size at most 2^n for the same language. This bound is tight.*

In particular the automaton in the example above requires an exponential number of states to be made deterministic.

The closure properties of automata are of particular importance. Indeed languages accepted by automata are closed under union (using a disjoint union construction) and intersection (using a product construction). They are also closed under reversal. The *reversal* operation consists of inverting the order of letters in all words. This is obtained by reversing the transitions and exchanging initial and final states. All these operations are polynomial (in particular the resulting automaton has polynomial size). However, the complement operation requires an exponential blowup in the number of states; in general one cannot do better than putting the automaton in deterministic form and then complementing.

The situation slightly differs for deterministic (and complete) automata. Intersection and union are still polynomial operations (this time union requires a product construction rather than a disjoint union). This time complement is easy (one just has to complement the set of final states), but reversal requires an exponential blowup.

3 Unambiguous forms of automata

The notion guaranteeing the unicity of the runs for each input is classically called unambiguity. In this section, we develop several notions of unambiguity, classical unambiguity in Section 3.1, and then strong unambiguity in Section 3.2. We continue our study with the motivating example of register automata (also called finite-memory automata) in Section 3.3. We conclude our description of this notion with the study of unambiguous automata over infinite words, and in particular with prophetic Büchi automata in Section 3.4 and over infinite trees in Section 3.5.

3.1 Unambiguity

A key consequence of the notion of determinism is that if an input is accepted, then there exists one and one only run which witnesses this acceptance. This yields the first notion of unambiguity, which can be applied to many models of automata.

▶ **Definition 2.** A (non-deterministic) automaton is *unambiguous* if on every input there is at most one accepting run.

Of course all deterministic automata are unambiguous (in any model of computation). Thus, on every model of computation enjoying determinisation, unambiguous automata are not less expressive than non-deterministic automata. The converse is not true, and some unambiguous automata are not deterministic, even on finite words. Consider for instance the automaton from the previous section recognising the language of finite words such that 'the n'th letter from the end is a b'. This automaton is co-deterministic, and as a consequence unambiguous, but it is not deterministic.

One can also remark that it is easy to decide (in polynomial time) whether or not an automaton is unambiguous. For this, given a non-deterministic automaton $\mathcal{A}$, one constructs another non-deterministic automaton $\mathcal{B}$ which accepts an input if and only if the original automaton had two distinct accepting runs over this input. This new automaton is essentially obtained by a product construction. It simulates concurrently on the input two instances of the original automaton. An extra gadget bit is used to detect that the two simultaneous runs differ at some moment. Of course, this new automaton accepts some word if and only if the original automaton is ambiguous. Since emptiness is decidable, we obtain that unambiguity is decidable (even in polynomial time). This argument is generic, and can be applied to most forms of automata. This is in particular true for all models of automata encountered in this document, and even beyond.

As one may expect, unambiguous automata have some algorithmic advantages compared to general non-deterministic automata. For instance, the problem of universality (whether an automaton accepts all inputs), the problem of inclusion and the problem of equivalence are PSPACE complete for languages described by non-deterministic automata [29]. The situation is different for unambiguous automata.

▶ **Theorem 3** (Hunt and Stearns [28])**.** *Over finite words, the problems of universality, inclusion and equivalence of languages accepted by unambiguous automata are polynomial.*

Such results suggest that the notion of unambiguity could be used in decision procedures in replacement for determinism. However, even if succinct unambiguous automata exist (more succinct than equivalent deterministic automata), one does not know how to efficiently construct them:

▶ **Question 1.** *Is is possible to efficiently disambiguate non-deterministic automata? In other words, does there exist a construction which, given a non-deterministic automaton, produces an unambiguous automaton of minimal size (or close to the minimal) for the same language, in time polynomial (in the sum of sizes of the input and the output)?*

Closure properties. Some closure properties are elementary to perform on the languages accepted by unambiguous automata. This is, in particular, the case for closure under union and intersection. In order to achieve this, in either case one simply performs a product construction. In contrast to deterministic automata, the languages of unambiguous automata are closed under reversal in linear time (the procedure for non-deterministic automata naturally preserves unambiguity). Note, once more, that these approaches are very generic, and that these closure properties are enjoyed by most models of unambiguous automata, and in particular all the models of automata appearing in this survey.

Another interesting operation on automata is *the cascade* (it is an operation on automata, not on languages). The principle of cascading two automata is to run the first automaton on the input, and then run a second automaton with the run of the first automaton as input.

This operation is easy to perform using again a product construction. It can be used to perform composition of transducers, and also generalises the construction of intersection and union. It is very useful for things such as composing operators in a temporal logic. What is interesting is that the cascade of non-deterministic automata yields a non-deterministic automaton, the cascade of deterministic automata yields a deterministic automaton, and the cascade of unambiguous automata yields an unambiguous automaton. The cascade operation is a generic operation which can be applied to most models of automata, including all the models of automata appearing in this survey. Cascade products preserve unambiguity in all cases. Over words, this operation allows us to easily construct complex unambiguous automata by cascading deterministic and co-deterministic automata.

However, it is erroneous to think that unambiguous automata are 'easy' to complement in the manner of deterministic automata. Indeed, the notion of unambiguity does only refer to how the automaton accepts an input, and it says nothing about rejecting an input.

▶ **Question 2.** *What is the state complexity of complementing an unambiguous finite word automaton? In particular, is it the case that the complement of the language of an unambiguous word automaton is accepted by an unambigous word automaton of polynomial size?*

Remark in particular that determinising an unambiguous automaton may yield an exponential blowup of the number of states, and thus it does not answer Question 2.

3.2 Strong unambiguity

To circumvent the problem of closure under complement, another notion of unambiguity can be used. We call it *strong unambiguity*.

▶ **Definition 4.** A *strongly unambiguous* (and complete) automaton is an automaton which can be used both as an unambiguous automaton for the language and as an unambiguous automaton for its complement.

There are several ways to implement this definition. The simplest way is to consider the couple of two unambiguous automata, one for the language, the other for the complement. Some more compact presentations can be used, for instance using a single automaton and (in the case of finite word automata) two couples of sets of initial states and final states, one to be used for accepting the language, the other for accepting the complement. It is easy to see that in terms of size, the two codings are similar up to a linear factor in size.

Note that testing if a pair of automata represent a strongly unambiguous automaton is doable in polynomial time, at least over words. Indeed, it involves checking the unambiguity of both automata, the emptiness of their intersection, and the universality of the union. All these tests can be performed in polynomial time, the last one using Theorem 3.

Closure properties. Strongly unambiguous automata are naturally closed under union, intersection, and cascade (using product constructions), and under complement. Strongly unambiguous automata are also closed under reversal in an easy way. Once more these properties are generic, and apply to most models of automata, including all of those encountered in this survey.

A natural question is the relationship between strong unambiguity and unambiguity in the case of finite words, and in particular the possible 'equivalence' of the two notions. The question is equivalent to Question 2.

▶ **Question 3.** *Is it possible to transform an unambiguous automaton into a strongly unambiguous one of polynomial size?*

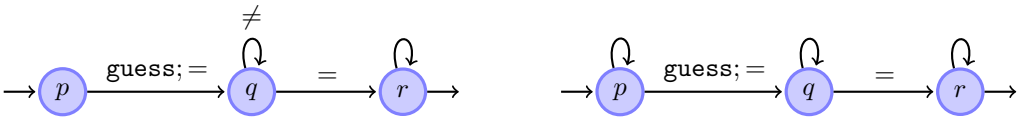
The two notions of unambiguity are interesting for finite word automata for complexity reasons. However, it is when dealing with other models of automata that the subtlety of the notion is revealed. We consider in the remainder of the section the case of register automata in Section 3.3, the case of Büchi automata over infinite words in Section 3.4 and finally the case of infinite trees in Section 3.5.

3.3 The motivating example of register automata

In this section, we show that notions of unambiguity, here strong unambiguity, can be used for defining classes of languages with nice properties. This requires a change of model, and going beyond classes of standard regular languages of finite words. We consider here the case of register automata. In this case, the notions of deterministic, unambiguous and non-deterministic automata have different expressive power. In particular the non-deterministic model is too strong (e.g., it is not closed under complement, and universality is undecidable). In this context it is interesting to consider the class of strongly unambiguous register automata; it generalises deterministic register automata, has good closure properties (in particular under complement and reversal contrary to deterministic automata), but does not inherit the bad decidability properties of the general class of non-deterministic register automata (universality is decidable).

A *data word* is a word over the alphabet $\mathbb{A} \times \mathbb{D}$ where $\mathbb{A}$ is a standard finite alphabet of *letters*, and $\mathbb{D}$ is an infinite alphabet of *data values*. A *register automaton* (variant of the model introduced by Kaminski and Francez [14] studied by Kaminski and Zeitlin [15] under the name 'finite-register automata with non-deterministic reassignement') has a finite number of *states*, and a finite number of *registers*, say $1, \dots, k$. Registers can store data values. A configuration of the automaton is a tuple $(p, d_1, \dots, d_k)$ consisting of a state and a data value for each register. At the beginning of the run, the automaton starts with any data value stored in its register (non-deterministic choice). Then the run proceeds as for a non-deterministic automaton whith reading the input, and at each time the transition describes how to use the data values of the register. Transitions are of the form $(p, a, u_1, \dots, u_k, q)$ where p and q are states, a is a letter (from $\mathbb{A}$), and $u_1, \dots, u_k$ are sequences of actions taken from $\{=, \neq, \texttt{guess}\}$. Such a transition goes from state p to state q when reading letter a and performs the sequence of actions on each counter as follows: action $=$ checks that the value of the register coincides with the data value read currently (otherwise it is not allowed to take the transition), the action $\neq$ is similar, but checks that the data values are different, and finally, the action `guess` possibly changes the value of the register, choosing the new value non-deterministically.

Let us give two examples of register automata. Here we assume that $\mathbb{A}$ contains only one letter, and we omit it wen drawing transitions. Furthermore, we consider only automata with one register. Thus transitions are simply labeled with sequence of actions.



The first one guesses a value and immediately checks that the guess coincides with the left-most data value in the word. Then, it compares the value of this register with all other positions in the word until it finds another occurrence of this value. In order to reach the final state, the value stored in the register during the first step should be encountered a second time in the word. It accepts the language 'the first data value reappears'. The second

automaton chooses non-deterministically a position in the word, stores its value, then chooses non-deterministically a second position in the word, and checks that the value is equal to its register. It accepts the language 'some data value appears twice'. The reader can check easily that the complement of the first language, namely the language 'the first data value does not occur elsewhere' is also accepted by a register automaton (set the set of final states to $\{p, q\}$). However, it is also well known that the complement of the second language, namely the language 'all data values are different' is not accepted by any register automaton.

Using direct adaptations of the constructions for classical word automata, one can show that the languages accepted by register automata are closed under union (using disjoint union of the automata) and intersection (using a product construction, with a set of registers containing all registers of the automata), and that their emptiness is decidable. The register automata are also closed under reversal of languages (this is not the case for the original model of Kaminski and Francez). However, on the negative side, register automata are not closed under complement, and their universality problem is undecidable. Overall, these automata seem to be too expressive, and it is relevant to search for sub-models which would enjoy a better balance between expressiveness and decidability.

A first worthwhile restriction to consider is deterministic register automata. A *deterministic register automaton* is a register automaton for which it is possible to construct in a unique way a run, step by step, while reading the input from left to right. For this, one requires that (a) there is exactly one initial state, (b) for each pair of distinct transitions reading the same letter from the same state, there exists a register whose action starts with $=$ in one of the transitions, and starts with $\neq$ in the other transition, (c) each `guess` action is immediately followed with an $=$ action, and (d) by convention the register share a common data value at the beginning of the run[1], say $\perp \notin \mathbb{D}$. The sequence of actions '`guess`;$=$' is called `store` since it amounts to collect the current data value, and store it in the register[2]. For instance, the first example of a register automaton above is in fact a deterministic register automaton. Deterministic register automata are closed under union, intersection, complement and cascade, and emptiness and universality are decidable. However, these automata lack the closure under reversal.

The notion of a *strongly unambiguous register automaton* is the one presented for finite word automata, instantiated for register automata. As in the usual case, the notion of strong unambiguity is semantic (in terms of existence of runs), but it is nevertheless a decidable property using the argument from the previous sections. Also, using the generic constructions described above, one easily shows that strongly unambiguous register automata are closed under intersection, union, complement and reversal. Furthermore, emptiness and universality are decidable. Hence, this model seems a good compromise between expressivity (it is more general than deterministic and co-deterministic automata) and decidability (most key properties of closure and decidability hold).

However, there is another way to define a class with similar properties. Consider the class of languages that are accepted by a register automaton and such that the complement

[1] Strictly speaking, this is not a restriction of the syntax of non-deterministic register automata, nevertheless, one easily shows that such deterministic register automata, up to minor modifications, can be seen as special cases of non-deterministic ones.

[2] The original finite-memory automata of Kaminski and Francez are allowed to use the action `store` but not the action `guess`. This restriction yields a weaker model, which is in particular not closed under language reversal. For instance the language 'the last data value does not appear before' cannot be recognised even with a non-deterministic finite-memory automaton. Kaminski and Zeitlin have studied more recently the model of 'finite-memory automata with non-deterministic reassignment' [15]. This model corresponds to register automata, where 'non-deterministic reassignment' stands for '`guess`'.

is also accepted by a register automaton. Since non-deterministic register automata are closed under intersection, union and reversal, the same holds for this class. Furthermore, the emptiness and universality of this class are decidable. However, this class has one important defect: it is not possible to decide if a pair $(\mathcal{A}, \mathcal{B})$ of register automata represents a valid language of the class. Indeed, it would require to decide if the union of the languages of $\mathcal{A}$ and $\mathcal{B}$ contains all words. This is the universality problem, which is undecidable for register automata.

The conjecture is that the two classes coincide.

▶ **Conjecture 4.** *The class of data languages that are accepted by register automata and whose complements are accepted by register automata coincide with the class of data languages accepted by strongly unambiguous automata.*

One direction is straightforward. Indeed, since the class of strongly unambiguous data languages is closed under complement, every language accepted by a strongly unambiguous automaton is accepted by a register automaton as well as its complement.

Something which may seem surprising is that the above conjecture may even be effective. This may happen in particular if it is proved as a consequence of the following conjecture.

▶ **Conjecture 5.** *Given two data languages K, L accepted by register automata and of empty intersection, there exists effectively a language U accepted by a strongly unambiguous register automaton such that $K \subseteq U$ and $L \subseteq \complement U$.*

This conjecture would imply Conjecture 4. Indeed, if K and L are complement of each other and both are accepted by a register automaton, then $U = K$ where U is the strongly unambiguous language from Conjecture 5. However, even if U is known, this would not help for deciding whether $K = \complement L$.

Another conjecture regarding this class reads as follows.

▶ **Conjecture 6.** *Strongly unambiguous register automata are equivalent to cascades of deterministic and co-deterministic register automata.*

Let us remark that the notion of cascade requires a bit more precision in this case. Indeed, cascading automata means executing a second automaton with a run of the first automaton as input. However, the question arises whether the second automaton, when reading the run of the first automaton, has access to the content of the registers of the first automaton during its run (as if these values were data-values written on the word). In the above conjecture, we assume that cascade allows this.

We terminate with register automata by remarking that Conjectures 4 and 5 make sense for the natural extension register automata to finite trees, while Conjecture 6 would have no obvious meaning in this context.

3.4 The case of infinite words: prophetic Büchi automata

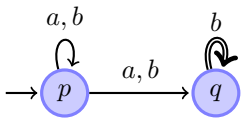
We have seen that it was straightforward for (finite) word automata to transform them into unambiguous ones. Indeed, it is sufficient to determinise. The situation for infinite words is different. We are dealing here with infinite words of length ω; *i.e.* words of the form $a_1 a_2 \dots$, with $a_1, a_2, \dots$ in a given finite alphabet $\mathbb{A}$. The set of words over the alphabet $\mathbb{A}$ is denoted $\mathbb{A}^\omega$.

A (non-deterministic) *automaton over infinite words*[3] $(Q, \mathbb{A}, \Delta, \mathcal{W})$ has a set of initial states I, an input alphabet $\mathbb{A}$, a transition relation Δ, and an accepting condition $\mathcal{W}$. The *accepting condition* $(\mathbb{C}, W)$ consists of an alphabet $\mathbb{C}$ and a language of infinite words $W \subseteq \mathbb{A}C^\omega$. The *transition relation* is a subset of $Q \times \mathbb{A} \times \mathbb{C} \times Q$. A *run* of such an automaton over a word $a_1a_2\dots$ is a sequence:

$$(q_0, a_1, c_1, q_1), (q_1, a_2, c_2, q_2), \dots$$

of transitions in Δ. It is *accepting* if $q_0 \in I$ and $c_1c_2 \cdots \in W$. A word is *accepted* if there exists an accepting run over it. The set of accepted words is the *language of the automaton*. An automaton over infinite words is called *determinisitic* (and complete) if for all input letter a and all states p, there exists one and only one transition of the form (p, a, c, q).

Automata over infinite words are classified according to their accepting condition. A *Büchi automaton* is an automaton over finite words using the accepting condition $\mathcal{B} = (\{1, 2\}, W_B)$ where W_B is the language of infinite words that contain infinitely many occurrences of letter 2. Transitions labelled with the condition letter 2 are called *Büchi transitions*. Said differently, a run in a Büchi automaton is accepting if it visits infinitely often some Büchi transition. Consider for instance the following Büchi automaton (where we take the convention that Büchi transitions are drawn as double arrows):



This automaton accepts the language of infinite words over $\{a, b\}$ that contain finitely many occurrences of the letter a. Indeed, for this automaton to accept an infinite word, the accepting run has to witness infinitely many occurrences of the Büchi transition from q to q. This means that the word has to eventually contain only b's. Conversely, given an infinite word with finitely many b's, one can construct an accepting run by starting in state p, and going to q when the last occurrence of letter a has been seen. This Büchi automaton is not deterministic. It is well known that this language is not accepted by any deterministic Büchi automaton (one interesting aspect of Büchi's seminal work was to be able to complement Büchi automata without determinising them [3]).

What McNaughton has later shown is that, using a stronger acceptance condition, it is possible to determinise Büchi automata. A *Müller accepting* condition (C, M) is such that M is a Boolean combination of properties of the form 'the letter c occurs infinitely often' [22].

▶ **Theorem 5** (McNaughton [20]). *A language of infinite words is accepted by a Büchi automaton if and only if it is accepted by a deterministic Müller automaton.*

A consequence of Theorem 5 is that Müller automata can be made deterministic (one has to remark that Müller automata are easy to transform into Büchi automata), and hence unambiguous. However, the question remains for Büchi automata: is it possible to transform Büchi automata into equivalent unambiguous ones? Prophetic automata positively answer this question. Indeed, every Büchi automaton can be made prophetic (Theorem 7), and every prophetic automaton is strongly unambiguous.

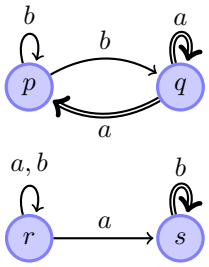
[3] We adopt here a terminology where accepting conditions label transitions rather than states. This is not the standard definition. However the various models are equivalent, and all the content of this section would be valid for the other model.

Prophetic automata are a strong form of co-deterministic automata; *i.e.*, automata such that for all states q and all letters a there exists at most one transition of the form (p, a, c, q). However, co-determinism (as studied in [1]) is not sufficient for guaranteeing unambiguity. Indeed, since the word is infinite, there can be several infinite runs over the same input even if it is co-deterministic (the proof that co-determinism entails unambiguity for finite words involves an induction on the length of the word which cannot be performed for infinite words).

▶ **Definition 6.** A Büchi automaton is *prophetic*[4] if it is universal and unambiguous when all its states are set to be initial.

So in particular if the prophetic automaton is such that all states are the origin of an accepting run over some input (useless states are removed), the unambiguity assumption entails that the automaton is co-deterministic. By definition, prophetic automata are strongly unambiguous.

For instance, consider the following Büchi automaton (without initial states):



For all words (over $\{a, b\}$) this automaton has exactly one accepting run (assuming all states are initial). Indeed, if a word has infinitely many a's, it is accepted only from state p if the first letter is b, and only from state q if its first letter is a. The word b^ω is accepted only from state s, and the words with finitely many a's (at least one) are accepted only from state r. Hence this automaton is prophetic. If one sets $\{r, s\}$ to be the set of initial states, the automaton accepts the language of words with finitely many occurrences of letter a. As explained above, this language is accepted by a non-deterministic Büchi automaton, but by no deterministic Büchi automaton. This prophetic automaton shows that the language is nevertheless accepted by a strongly unambiguous Büchi automaton.

Though the subject of this survey is not to dig further in this notion of prophetic automata, let us show that it has some quite intriguing properties. In particular, these automata are in some sense 'minimal in the long run'. This can be illustrated by the following remark: 'there is exactly one prophetic automaton (that has only productive states) for the full langage over the unary alphabet $\{a\}$, and this automaton has only one state'. This property is very different from what we are used to with deterministic automata. Indeed, there are infinitely many deterministic Büchi automata for this language.

The key theorem concerning prophetic Büchi automata is that such automata exist for all regular languages of infinite words.

▶ **Theorem 7** (Michel and Carton [5]). *For all regular languages of infinite words, there exists an equivalent prophetic automaton.*

[4] The reader should be aware that prophetic are called 'unambiguous automata' in [5]. We avoid this terminology here which would be strongly ambiguous.

A direct consequence of this result is that, in particular, Büchi automata can be made strongly unambiguous.

This is not the only interest of prophetic automata. Another motivation is the use of prophetic automata for the characterisation and decidability of classes of regular languages. In general, one is interested in questions of the form:

> Does a given regular language L belong to a fixed sub-class of regular languages?

This kind of questions was initiated with the famous seminal work of McNaughton-Papert-Schützenberger [27, 21] characterising in an effective way the languages of finite words that can be defined in first-order logic. There is now a rich variety of results of this form. The decision procedure for such results usually starts by assuming that the language L is represented in a specific suitable form. For instance, to characterise first-order logic, one should start with a minimal deterministic automaton, or with the syntactic monoid. For deciding the star-height one starts with another form of automata [16, 7]. We will also see in Section 4 guidable automata which were introduced for attacking the Mostowski hierarchy problem. In general, it is an interesting high level question to be able to relate the properties of the class of regular languages one aims at with the suitable form of representation needed for the language.

Prophetic automata enter this picture when one is interested in characterising the regular languages of infinite words corresponding to some temporal logics, and more specifically, if one wants to answer the following question:

> Is it possible to define a regular language of infinite words L in temporal logic using only the operator `neXt` (*resp.* `Until`)?

This problem is decidable [24], and the representation of the language used in the proof and in the decision procedure is a prophetic automaton for L.

There is an intuitive reason for that: the temporal logics in consideration only use future modalities, and thus nicely cohabit with prophetic automata. One way to witness this nice cohabitation is to remark that a formula of such a temporal logic is naturally encoded as a cascade of elementary prophetic automata representing the operators of the logic. This would not hold if the logic could use past operators. Nor if automata would be deterministic rather than prophetic.

3.5 The case of infinite trees and inherent ambiguity

We have only encountered word automata so far. The theory of automata over trees is also very rich (see e.g., [10] for an extensive presentation).

For simplicity, we will use a slightly restrictive notion of trees. This is solely for simplifying the technical aspects of the discussion. We will consider binary *tree*s, with labels (from a finite alphabet, say $\mathbb{A}$) only on inner nodes (*i.e.*, nodes that are not leaves). Each node has either two children, or it is a leaf. One denotes by ε the root node of the tree, and given an inner node x, one denotes by $x0$ its left child and by $x1$ its right child. The *label* at inner node x in tree t is denoted $t(x)$. One finally denotes by $nodes(t)$ the set of nodes of the tree t. A tree is finite if it has finitely many nodes, otherwise it is called infinite. We call the *infinite binary tree* the only infinite tree without leaves over a unary alphabet.

A *finite tree automaton* $(Q, \mathbb{A}, I, \Delta, F)$ has a finite set of *states* Q, an *input alphabet* $\mathbb{A}$, a set of initial states I, a *transition relation* $\Delta \subseteq Q \times \mathbb{A} \times Q \times Q$, and a set of *final states* F. A *run* (ρ, δ) over a tree t is a mapping ρ from nodes of t to Q and a mapping δ from inner nodes to Δ such that for every inner node x, $\delta(x) = (\rho(x), t(a), \rho(x0), \rho(x1))$. The run is

initial if $\rho(\varepsilon)$ is initial. The run is *final* if $t(x) \in F$ for all leaves x. The run is *accepting* if it is both initial and final. Note that ρ is sufficient by itself for defining the run. The mapping δ will become necessary when we consider infinite trees.

Over finite words, determinism and co-determinism played a symmetric role, and so had exactly the same properties. This is not the case over trees, due to the absence of similar symmetry. For this reason, one distinguishes two different notions of determinism. A tree automaton is *bottom up deterministic* (and complete) if (a) it has exactly only one final state and (b) for all states q, r and letter a, there is exactly one transition of the form (p, a, q, r). A tree automaton is *top-down deterministic* (and complete) if (a) it has only one initial state, and (b) for all state p and all letters a there exists exactly one transition of the form (p, a, q, r). A top-down deterministic automaton has exactly one initial run on each input. A bottom-up deterministic automaton has exactly one final run on each *finite* input (there can be several runs over infinite trees).

▶ **Theorem 8.** *Automata over finite trees can be made bottom-up deterministic, but not top-down deterministic in general.*

The fact that every tree automaton can be made bottom-up deterministic is obtained by a powerset-construction similar to the determinisation of word automata. For the impossibility of obtaining top-down deterministic automata it is witnessed by the following very simple tree language:

$$\left\{ \begin{array}{ccc} & \mathrm{a} & \\ \mathrm{a} & & \mathrm{b} \end{array} \;,\; \begin{array}{ccc} & \mathrm{a} & \\ \mathrm{b} & & \mathrm{a} \end{array} \right\} .$$

Indeed, a top-down deterministic automaton, when processing the tree, has to guess whether the b-subtree will occur in the left or the right sub-tree. This contradicts determinism. Formally, the proof shows that if a top-down deterministic automaton accepts the above trees, then it accepts also the 'three-a's' tree.

Over finite trees it is possible to provide a strongly unambiguous automaton by using bottom-up deterministic automata according to Theorem 8. But what about infinite trees?

Formally an infinite tree automaton is defined as for finite trees, but is further enhanced with an accepting condition $\mathcal{W} = (\mathbb{C}, W)$ (as defined in Section 3.4), and each transition is of the form (p, a, c, q, r) where c is an extra component from $\mathbb{C}$. A run ρ over an infinite tree is now called *final* if all its leaves belong to F, and furthermore for every infinite branch $x_0, x_1, \ldots$, the word $\rho(x_0)\rho(x_1)\cdots \in W$. A run is *accepting* if it is both initial and final. A Müller automaton for infinite trees is such an automaton which uses a Müller acceptance condition. A language of infinite trees is called *regular* if it is the language of some Müller automaton over infinite trees. Regular languages of infinite trees are known to be closed under union, intersection, and complement (a famous result of Rabin [25]).

For instance, consider the Büchi automaton for infinite trees over the alphabet $\{a, b\}$ that has states $\{p, q\}$, initial state p, final state q, and transitions:

$$\{\, (p, b, 1, p, \top), (p, b, 1, \top, p), (p, a, 1, \top, \top), (\top, a, 2, \top, \top), (\top, b, 2, \top, \top) \,\} .$$

The language of this automaton is the language 'there is an occurrence of the letter a'. Observe first that all trees are accepted from state $\top$. The state p should be understood as 'searching for letter a'. Given an input, the automaton has to start at root position in state p; *i.e.*, it 'searches for letter a'. Then the state p is propagated as long as a letter b is encountered (since only the two first transitions can be used from p while reading b). Each

time the automaton has to decide to pursue its search to the left subtree (first transition) or to the second subtree (second transition). When finding an occurrence of a, there is nothing more to check and the third transition is used. Finally, the Büchi condition guarantees that the two first transitions are not used infinitely often along an infinite branch; *i.e.*, it prevents the automaton from searching, never finding, but still accepting the input infinite tree.

▶ **Theorem 9** (Carayol, Löding, Niwinski and Walukiewicz [4]). *The language of infinite trees 'there is an occurrence of the letter a' is* intrinsically ambiguous*; i.e., there exists no unambiguous automaton for this language.*

The proof of this result is by reduction to the result of the non existence of a regular choice function over the infinite binary tree.

In order to state the result concerning choice one needs to define the regular relations over trees. Firstly, given $U_1, \ldots, U_k$ subsets of the nodes of t, denote by $t[U_1, \ldots, U_k]$ the tree t in which the label of node x is set to $(t(x), b_1(x), \ldots, b_k(x))$ where $b_i(x) = 1$ if $x \in U_i$ and 0 otherwise, for all i. In other word $t[U_1, \ldots, U_k]$ is the tree t enhanced with the characteristic functions of the sets $U_1, \ldots, U_k$ as extra labelling information. Call now a relation $R \subseteq (2^{nodes(t)})^k$ *regular* if there is an automaton which accepts $t[U_1, \ldots, U_k]$ if and only if $(U_1, \ldots, U_k) \in R$. In other words, there exists an automaton which, when given the various input sets as extra labelling of the tree, is able to check whether the relation holds between them. We will use this definition for a function from sets of nodes to nodes. Such a function can be seen in the usual way as a relation between sets and singleton sets.

A *choice function* on a tree t is a function which maps each non-empty subset of nodes of t to one of its elements; it chooses an element in the set given as input.

▶ **Theorem 10** (Gurevich and Shelah [12], see [4] for a simple proof). *There is no regular choice function on the infinite binary tree*[5].

The phenomenon raised by this theorem is that an automaton is able to test that a set of nodes is non-empty, but it is unable to pinpoint in a unique way an element witnessing this non-emptiness. The reduction essentially says that if an unambiguous automaton for the language 'there exists an occurrence of letter a' existed, then this automaton would have to identify in some way a specific occurrence of the letter a as witness[6]. Thus it could be seen as choosing some occurrence of a inside the set of occurrences of a. In other word, this automaton could be used, after some slight modifications, as an implementation of a choice function. This would contradict Theorem 10.

4 Guidable automata

A guidable automaton is an automaton over finite or infinite trees which has some properties that resemble those of a top-down deterministic automaton. Recall that top-down deterministic automata are not as expressive as general automata, already over finite trees. The notion was introduced in [8] and is deeply studied in [17].

Intuitively, a guidable automaton is an automaton which requires the minimum quantity of information for resolving its non-determinism. This informal definition is stated as 'given any

[5] The theorem of Gurevich and Shelah is stated in terms of definability in monadic second-order logic. Thanks to the result of Rabin [25], the statements are equivalent.

[6] For instance, the above automaton for the language 'there is an occurrence of letter a' witnesses letter a, namely when it uses the transition $(p, a, \top, \top)$. But this automaton is ambiguous, and for this reason there may be several positions on which this transition could be used.

accepting run of any automaton for the same language, it contains sufficient information for resolving the non-determinism.' Another supporting intuition is that a guidable automaton is able to 'simulate' the behaviour of any automaton for this language.

To implement this, let us first describe what is a *guide* from an automaton $\mathcal{A}$ to an automaton $\mathcal{B}$. A guide is an application:

$$g : \quad Q_{\mathcal{B}} \times \Delta_{\mathcal{A}} \to \Delta_{\mathcal{B}} \ ,$$

where $Q_{\mathcal{B}}$ are the state of $\mathcal{B}$, and $\Delta_{\mathcal{A}}$ and $\Delta_{\mathcal{B}}$ are the set of transitions of the automata $\mathcal{A}$ and $\mathcal{B}$ respectively.

Let us show now how to extend a guide g into a function $\tilde{g}$ from the runs of $\mathcal{A}$ to runs of $\mathcal{B}$. Assume that the automaton $\mathcal{B}$ has a unique initial state q_0, and consider some run $(\rho_{\mathcal{A}}, \delta_{\mathcal{A}})$ of $\mathcal{A}$ over some input tree t. One constructs a run for $\mathcal{B}$ inductively as if g was defining a top-down deterministic transducer of states $Q_{\mathcal{B}}$, reading the run of $\mathcal{A}$. Then there is at most one run (ρ, δ) of $\mathcal{B}$ such that:

$$\rho(\varepsilon) = q_0 \qquad \text{and} \quad \delta(x) = g(\rho(x), \delta_{\mathcal{A}}(x)) \quad \text{for all inner nodes } x \ .$$

This initial run, if it exists, is called the run of $\mathcal{B}$ driven by $(\rho_{\mathcal{A}}, \delta_{\mathcal{A}})$.

An infinite tree automaton $\mathcal{A}$ *guides* an infinite tree automaton $\mathcal{B}$ if there exists a guide g such that for every accepting run ρ of $\mathcal{A}$, $\tilde{g}((\rho, \delta))$ is an accepting run of $\mathcal{B}$ over the same infinite tree. A direct consequence of this is that the language of $\mathcal{A}$ is included in the language of $\mathcal{B}$. In fact, the guide g can be understood as an 'easy to verify' certificate for this inclusion.

▶ **Definition 11.** An infinite tree parity[7] automaton $\mathcal{B}$ is *guidable* if for all infinite tree parity automata $\mathcal{A}$ generating a sub-language of $\mathcal{B}$, $\mathcal{A}$ guides $\mathcal{B}$.

Thus, an automaton is guidable if for all automaton generating a sub-language there is an easy certificate for the inclusion of the languages.

You can remark easily that every top-down deterministic automaton is guidable. Indeed, the guide uses the input transition from $\Delta_{\mathcal{A}}$ just for determinining the letter, and has no choice concerning the transition to take. The above infinite tree Büchi automaton for the language 'there is an occurrence of letter a' is guidable. Unfortunately, this would be too technical to establish here.

An important property of guidable automata is that they always exist.

▶ **Theorem 12** ([8]). *Every regular language of infinite trees is accepted by a guidable parity automaton.*

The construction requires a doubly exponential blowup, and this is tight [17].

This theorem was used for an attempt to deciding the Mostowski hierarchy for non-deterministic automata; *i.e.* the hierarchy induced by the number of priorities necessary for a parity automaton to accept a language of infinite trees. It seems that guidable automata are the correct form of automata to analyse in a procedure for deciding this hierarchy. For the moment the problem has only been reduced to another problem, the existence of a bound on the function computed by some forms of automata [8].

Löding has proposed a method for deciding if a given infinite tree parity automaton is guidable [17]. The key ingredient is to able to construct, given two infinite tree parity

[7] The parity conditions are special cases of Müller conditions: the letters are integers, and the accepting language contains those words such that the maximal integer encountered infinitely often is even.

automata $\mathcal{A}$ and $\mathcal{B}$, a finite Rabin game which is won by the first player if and only if $\mathcal{A}$ guides $\mathcal{B}$ (and in this case the positional winning strategy induces the guide). The algorithm for deciding if a given infinite tree parity automaton $\mathcal{B}$ is guidable is then to first transform $\mathcal{B}$ into a guidable automaton $\mathcal{A}$ (this requires a doubly exponential blowup in the number of states) and then check that $\mathcal{A}$ guides $\mathcal{B}$. This is a doubly exponential procedure.

▶ **Question 7.** *Can we decide efficiently if a infinite tree parity automaton is guidable? If the answer is no, does there exist small certificates that an infinite tree automaton is guidable? What is the state complexity of complementing a guidable automaton?*

5 History-determinism

5.1 Principle

History-determinism is a notion suitable for automata running over words (finite or infinite). It has been introduced for its use in the context of games of infinite duration by Henzinger and Piterman [13] under the name *'good for solving games'*. The same notion was introduced independently in [6] for cost functions, and used together with Löding in [9] for developing the theory of cost functions over finite trees.

Given an infinite word automaton $\mathcal{A} = (Q, \mathbb{A}, I, \Delta, \mathcal{W})$ (the automaton may have infinitely many states) and a mapping τ from Q to some set Q', denote by $\tau(\mathcal{A})$ the *homomorphic image* of $\mathcal{A}$ by τ, that is the automaton $(Q', \tau(I), \tau(\Delta), \mathcal{W})$ where $\tau(\Delta)$ is the transition relation

$$\{(\tau(p), a, c, \tau(q)) \ : \ (p, a, c, q) \in \Delta\} \ .$$

The homomorphic image of an automaton over finite words is defined in a similar way, omitting the accepting condition part, and replacing the final states F by $\tau(F)$.

Let us remark that $\mathcal{A}$ may be deterministic and $\tau(\mathcal{A})$ not be deterministic. Note also that the language of the homomorphic image is always a superset of the language of the original automaton. This simply comes from the fact that accepting runs in $\mathcal{A}$ have an accepting homomorphic image by τ. Note finally that it can be a strict superset.

▶ **Definition 13.** An automaton is *history-deterministic* if it is the homomorphic image of a (possibly infinite) deterministic automaton for the same language.

The idea behind the notion of history-determinism is that an automaton can be non-deterministic, but that one can use the deterministic automaton from which it is the homomorphic image as an oracle for resolving the non-deterministic choices. If an automaton is history-deterministic, and if adding a new transition or a new initial state to it does not change the language, then the obtained automaton is again history-deterministic.

Consider a finite or infinite word history-deterministic automaton $\mathcal{H}$ which the homomorphic image $\tau(\mathcal{A})$ of a deterministic automaton for the same language. For p a state of $\mathcal{A}$, call A_p the language accepted from state p by $\mathcal{A}$. For q a state of $\mathcal{H}$, define in the same way H_q the language accepted by $\mathcal{H}$ from state q. We claim that for all states p of $\mathcal{A}$ reachable from the initial state, $A_p = H_{\tau(p)}$. One direction is straightforward. Indeed, any accepting run of $\mathcal{A}$ from state p is send by τ to an accepting run over the same input of $\mathcal{H}$ from sate $\tau(p)$. Conversely, assume a word u is accepted by $\mathcal{H}$ from state $\tau(p)$. Let w be a run such that $\mathcal{A}$ reaches state p after reading it (reachability assumption). This means that wu is accepted by $\mathcal{H}$. Thus by the history-determinism assumption, wu is accepted by $\mathcal{A}$. It follows, since $\mathcal{A}$ is deterministic, that u is accepted by $\mathcal{A}$ from state p. This proves the claim.

We directly deduce from it the following characterisation of history-deterministic automata over finite words.

▶ **Proposition 14.** *An automaton over finite words is history deterministic if and only if it contains a deterministic sub-automaton for the same language.*

Indeed the construction is as follows, where we reuse the above notations for $\mathcal{A}$ and $\mathcal{H}$. First of all, one can consider that all states of $\mathcal{A}$ are reachable. Otherwise one removes the non-reachable states, and consider that $\mathcal{H}$ is the homorphic image of this restricted automaton. Then one chooses as unique initial state $\tau(q_0)$. Finally one removes sufficiently many transitions from $\mathcal{H}$ for making it deterministic without creating a dead-end; *i.e.* without ever removing the last transition issued from a given state reading a given letter. One then checks that none of these edge removals change the language accepted by $\mathcal{H}$. This is obvious since whenever a transition has been removed from $\mathcal{H}$, say (p, a, q), then another transitions (p, a, q') was existing. But, by the above claim, (and since we have only kept the states which are homomorphic images of reachable states of $\mathcal{A}$), $H_q = H_{q'}$. It follows that removing (p, a, q) or (p, a, q') (but not both) from the automaton does not change its language.

It is natural to wonder whether this result can be extended.

▶ **Conjecture 8.** *A parity automaton is history-deterministic if and only if it contains a deterministic sub-automaton for the same language.*

Let us remark that in the case of finite words, history-determinism has an efficient decision procedure.

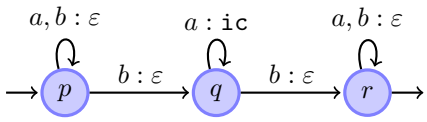
▶ **Theorem 15** (Löding [18])**.** *One can decide in polynomial time if an automaton has a deterministic sub-automaton for the same language.*

5.2 The motivating example of cost functions

Proposition 14 and Conjecture 8 tend to make us think that the notion is not very interesting. Indeed it implies that history-deterministic automata are always of bigger size than deterministic ones for the same language. So why would we be interested in using such forms of automata?

An enlightening example is to consider the case of cost functions. In this model deterministic automata are strictly weaker than non-deterministic ones, but history-deterministic automata are as expressive as non-deterministic automata.

Let us describe this model of automata through examples. Consider the following automaton:

$\mathcal{B} =$ 

This automaton is a *one counter B-automaton* ([6], following ideas from [16] and [2]). We do not want to enter a full description of this object. Let us simply describe its semantics. The interesting reader can refer to [9] for more precise definitions. In our case, the automaton possesses one counter, which ranges over non-negative integers. At the beginning of the run, the counter assumes value 0. Three actions can be performed on the counter:

- it can be left unchanged, using action ε,
- it can be incremented by 1, using action `ic`,

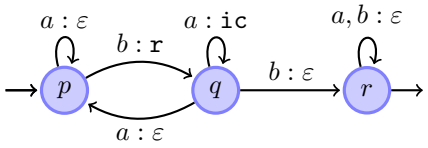
- it can be reset to 0, using action `r`.

Each transition is labelled $a_1, \cdots : \alpha$, where $a_1, \ldots$ are the letters read by the transition for being fired, and α is the action performed on the counter. A run of the automaton is defined as usual. A run is said *n-accepting* if it starts in an initial state, end in a final state and the counter never exceeds value n along the run (this can be seen as a run under a resource consumption constraint). The *n-language* accepted by the automaton is the set of words which are n-accepted. This first automaton n-accepts a word if and only if the input word contains a factor of the form ba^mb for some $m \leq n$. [8]

One needs to adapt the notion of history-determinism to B-automata. A B-automaton is *history-deterministic*[9] if for all n, there is a deterministic B-automaton of same n-language, the homomorphic image of which is $\mathcal{A}$. In other words, the automaton is unique, but should be uniformly history-deterministic for all acceptance conditions.

Let us give an intuitive meaning why the above automaton is not history-deterministic. Fix yourself some n, for instance 1. Now imagine that you feed this automaton with a word over $\{a, b\}$, letters by letters, yielding a word of value n. Your goal is to construct step by step an n-accepting run of the automaton. At the beginning of the run, you are in state p. While receiving each letter, you have to decide what transition to take. The non-determinism occurs when in state p, while reading letter b. Your problem is that you want to go to state q only when a segment of a's of length n is about to arrive next. Whatever is your computational power, if you do not know the letters to arrive next, you are unable to decide if should proceed to p or to q. This informal argument is the reason why this first B-automaton is not history-deterministic: it is impossible to resolve the non-determinism without looking the input ahead. This corresponds to the above notion of history-determinism since, assuming history-determinism, the automaton from which you are the homomorphic image is deterministic, and hence it has all the computation power possible (there is no size constraint, not even finiteness), but because of its determinism it has no access to the future of the input.

Consider now the following automaton:

$\mathcal{H} =$ 

Consider an n-accepting run of this automaton over the input, then it has to read a p at some point, and enters state q, then spends at most n steps in state p, and then reads again a b allowing to proceed to r which is the only accepting state. This means that the word contains a factor ba^mb for $m \leq n$. Let us show conversely how to construct 'in a deterministic way' an n-accepting run over a word containing a factor ba^mb for $m \leq n$. The run starts in state p. As long as a letter is met, and there is no non-deterministic choice, the run is prolongated using the only transition available. The only case of non-determinism is when the run is in state q and the input letter is an a. In this case, one has to choose to either stay in p–call this choice '*continue*'–or to go to p–call this choice '*skip*'–. One resolves this non-determinism by choosing to continue as long as the counter has a value smaller than n,

[8] One often sees B-automata as defining functions. The *function computed by a B-automaton* is the least n such that there exists an n-accepting run, or infinity if there is no such run. In this case, the automaton computes the least size of a block of consecutive a's surrounded by b's.

[9] The exact definition requires this modulo an approximation [6]. We do not enter these details here.

and to skip otherwise. Using this construction, if the input word contains a factor ba^mb for $m < n$, then the word has to terminate in state r which is accepting. This strategy of constructing the run can be implemented by a deterministic automaton which would know (a) what state the automaton $\mathcal{H}$ is in and (b) the current value of the counter. In this case this bigger deterministic automaton (it has $3(n+1)$ states) is sent by homomorphic image to $\mathcal{H}$ (by projecting out the counter value), and accepts the same n-language. This means that $\mathcal{H}$ is history-deterministic. Furthermore it is equivalent to $\mathcal{B}$.

In the above example, it is clear that it is required to know the current value of the counter. And in fact, it would be impossible to provide a determinisitic or even an unambiguous B-automaton which would accept the same n-language for all n. In this case, the notions of determinism and history-determinism completely differ in expressiveness.

The general theory is a bit more involved, since automata need be considered modulo an approximation $\approx$ in order to be made history determinisitic. We do not define it here. Nevertheless, the following statement should sustain the interest of the notion of history-determinism.

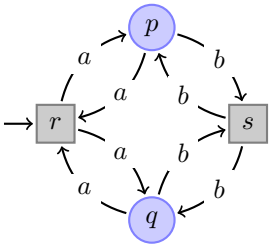
▶ **Theorem 16** ([6]). *B-automata cannot be made deterministic nor unambiguous, even up to $\approx$. Every B-automaton is equivalent to an history-deterministic one, up to $\approx$.*

In the last section, we explain why history-deterministic automata are good for solving games. This has been used in particular for proving results concerning cost functions in [9].

5.3 History-determinism is good for solving games

As mentioned above the key motivation for introducing history-determinism is its use for solving games. We will explain this for automata over infinite words, though the main application of the technique is for cost-functions.

We start by describing what is a game. A *game* $(A, \mathcal{W})$ consists of a *winning condition* $\mathcal{W} = (\mathbb{C}, W)$ (corresponding to an accepting condition in Section 3.4), and an *arena* A which is a (possibly infinite) directed graph with edges labeled by $\mathbb{C}$, and such that the vertices (called *positions*) are partitioned into positions for *Eva* and positions for *Adam*. Furthermore, there is a distinguished *starting position* p_0. Traditionally, the positions of Eva are denoted by circles, and the positions of Adam by squares. The following example



The game is played by two opponents, Eva and Adam. It starts by putting a token on the starting position position (here r). Then the token is moved by the owner of the position. Here Adam for the first move. The player chooses the edges along which the token should move. Here, either to p or to q. Then the game proceeds with the new position. In the above case, this is the turn of Eva to play. In the end, the two players will have constructed an infinite path in the arena, labeled with some infinite word w over $\mathbb{C}$. This play is *winning for Eva* if $w \in W$. Otherwise it is *winning for Adam*. A player is winning if he has a strategy specifying what he should play such that, whatever are the choice of the opponent, the

resulting play is winning for him. Formally, a *strategy* is defined as a mapping from partial plays (the prefix of a play), ending in a state of the player to edges, the transition taken by the strategy after having seen this prefix. In the above case, if the winning condition is the set of languages that contains infinitely many occurrences of both a and b, then Eva (circle) wins the game by alternatively playing left and right.

A winning condition is called *determined* if for all games using this winning condition, one of the players has a winning strategy. It is possible to construct winning conditions which are not determined. However, thanks to Martin's theorem [19], every reasonable accepting condition, and in particular every regular winning condition is determined. This is an important property since it allows us to reason with arguments such as if there is no strategy for one player, then the other has a winning strategy: it allows to complement an existential property 'there is a strategy for one player', and obtain again an existential property 'there is a strategy for the other player' ($\star$).

Games play an important role in automata theory since the seminal work of Gurevich and Harrington [11], and in particular in the proof of closure under complementation of the regular languages of infinite trees. More specifically, complementing a regular language means negating a property of the form 'there exists a run of the first automaton', and obtain a property of the form 'there exists a run of the complement automaton'. This is exactly what games are good for (cf. $\star$). But for this approach to work, the automata must be able to 'manipulate' the strategies. The issue is that the strategy is a very complex object. Indeed it tells what should be the next arrow to take knowing the full history of the play so far. This is too much information for an automaton. For this reason, the simpler *finite memory* strategies play an important role. In the above example, player Eva requires one bit of memory in order to switch between left and right, and be sure to win: this is a memory 2 strategy. In fact, the condition 'infinitely many a's and infinitely many b's' requires memory at most 2 in every game. Said differently, on every game with this winning condition, if Eva wins, then she needs only one bit of memory for implementing a winning strategy. More generally, every winning condition which is a regular language of infinite words requires only a finite amount of memory to implement the winner's strategy.

It would be too long to explain in mode detail why finite memory strategy are important for complementing infinite tree automata. Another advantage of the notion is that it helps for deciding the winner in a finite game. Indeed, it is sufficient to guess a finite memory strategy (and this is an object of linear size), and then check that this strategy is winning (often a polynomial task). This is how one proves that deciding the winner of a game over a fixed regular winning condition is in NP$\cap$coNP (it is more complex if the winning strategy is part of the input).

Now, consider a game $\mathcal{G}$ of winning condition $\mathcal{L} = (\mathbb{B}, L)$, and assume you have a deterministic infinite word automaton $\mathcal{A}$ for the language L of accepting condition $\mathcal{W} = (\mathbb{C}, W)$. Then, a natural thing to do is to compose the two objects. This yields a new game $\mathcal{G} \otimes \mathcal{A}$, the position of which are ordered pairs of a position in $\mathcal{G}$ and a state of $\mathcal{A}$, and of winning condition $\mathcal{W}$. At each turn of the game, the player chooses to move the token according to the first game, and the second component is updated according to the transitions of the automaton, outputting a letter from $\mathbb{C}$. Formally, a new position (p, q) belongs to the same player as p in $\mathcal{G}$, and there is an edge from (p, q) to (p', q') labelled c if there is an edge from p to p' labelled b in $\mathcal{G}$, and a transition of the form (q, b, c, q') in $\mathcal{A}$. The starting position is the pair of the starting position in $\mathcal{G}$ and the initial state from $\mathcal{A}$. It is quite clear that playing this new game amounts to play the original game, and update at the same time the state of the automaton. Since the automaton accepts the winning

condition $\mathcal{L}$, the winner is preserved.

▶ **Theorem 17.** *The games $\mathcal{G}$ and $\mathcal{G} \otimes \mathcal{A}$ have same winner. Furthermore, if the winner of $\mathcal{G} \otimes \mathcal{A}$ can win with memory k, then he/she can win with memory $k|\mathcal{A}|$ the game $\mathcal{G}$.*

The memory part of this statement is also natural: in order to translate a winning strategy from $\mathcal{G} \otimes \mathcal{A}$ to a winning strategy for $\mathcal{G}$, it is necessary to maintain (a) sufficient memory for winning the game $\mathcal{G} \otimes \mathcal{A}$, and furthermore (b) sufficient memory for keeping track of the state the automaton $\mathcal{A}$ should be in. Hence an upper bound of $k|\mathcal{A}|$.

A similar construction can be done, with $\mathcal{A}$ not deterministic. In this case, in the product game $\mathcal{G} \otimes \mathcal{A}$, player Eva is furthermore in charge of constructing the run of the automaton (this is implemented in the precise definition of the game). But, if $\mathcal{A}$ is not deterministic, what remains true is that if Adam wins $\mathcal{G}$, then Adam wins $\mathcal{G} \otimes \mathcal{A}$. The converse does not necessarily hold. Indeed, the game $\mathcal{G} \otimes \mathcal{A}$ requires Eva to resolve the non-determinism of $\mathcal{A}$. But, when Eva makes such a choice, Adam can take advantage of this information for winning the game. History-determinism is here the right notion, and this is why the notion is also called 'good for solving games' in [13]:

▶ **Theorem 18.** *If $\mathcal{H}$ is history-deterministic, then $\mathcal{G}$ and $\mathcal{G} \otimes \mathcal{H}$ have same winner.*

What differs here compared to the deterministic case is that, when translating the winning strategy for Eva in the game $\mathcal{G} \otimes \mathcal{H}$ to $\mathcal{G}$, one uses the automaton $\mathcal{A}$ from which $\mathcal{H}$ is the homomorphic image as second component.

The intriguing consequence of this is that the memory needed to win the game $\mathcal{G}$ is now $k|\mathcal{A}|$ and not $k|\mathcal{H}|$ as if $\mathcal{H}$ was deterministic. This is interesting since the automaton $\mathcal{A}$ may a priori be very large compared to $\mathcal{H}$. Using this construction, one reduces a game $\mathcal{G}$ which may require a lot of memory to a slightly larger game $\mathcal{G} \otimes \mathcal{H}$ of same winner which uses much less memory.

In the context of regular winning conditions, assuming Conjecture 8 holds, we cannot hope to really win something, since this means that one could choose the automata $\mathcal{H}$ and $\mathcal{A}$ of same size. In [9] the technique is used for cost functions. In this context, it allows one to transform games that require an unbounded quantity of memory into games which require no memory. History-deterministic automata play a crucial role in the theory of cost-functions for this reason.

Acknowledgment

I am very grateful to Christopher Broadbent, Olivier Carton, Christof Löding, Gabriele Puppis and Thomas Wilke for their discussion and their help during the preparation of this document.

References

1 Danièle Beauquier and Dominique Perrin. Codeterministic automata on infinite words. *Inf. Process. Lett.*, 20(2):95–98, 1985.

2 Mikolaj Bojańczyk and Thomas Colcombet. Bounds in ω-regularity. In *LICS 06*, pages 285–296, 2006.

3 J. Richard Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford Univ. Press, 1962.

4 Arnaud Carayol, Christof Löding, Damian Niwinski, and Igor Walukiewicz. Choice functions and well-orderings over the infinite binary tree. *Central European Journal of Mathematics*, 8(4):662–682, 2010.

5 Olivier Carton and Max Michel. Unambiguous büchi automata. *Theor. Comput. Sci.*, 297(1-3):37–81, 2003.

6 Thomas Colcombet. The theory of stabilisation monoids and regular cost functions. In *Automata, languages and programming. Part II*, volume 5556 of *Lecture Notes in Comput. Sci.*, pages 139–150. Springer, Berlin, 2009.

7 Thomas Colcombet and Christof Löding. The nesting-depth of disjunctive μ-calculus for tree languages and the limitedness problem. In *Computer science logic*, volume 5213 of *Lecture Notes in Comput. Sci.*, pages 416–430. Springer, Berlin, 2008.

8 Thomas Colcombet and Christof Löding. The non-deterministic Mostowski hierarchy and distance-parity automata. In *Automata, languages and programming. Part II*, volume 5126 of *Lecture Notes in Comput. Sci.*, pages 398–409. Springer, Berlin, 2008.

9 Thomas Colcombet and Christof Löding. Regular cost functions over finite trees. In *LICS*, pages 70–79, 2010.

10 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 2007.

11 Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *STOC*, pages 60–65, 1982.

12 Yuri Gurevich and Saharon Shelah. Rabin's uniformization problem. *J. Symb. Log.*, 48(4):1105–1119, 1983.

13 Thomas A. Henzinger and Nir Piterman. Solving games without determinization. In Zoltán Ésik, editor, *CSL*, volume 4207 of *Lecture Notes in Computer Science*, pages 395–410. Springer, 2006.

14 Michael Kaminski and Nissim Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

15 Michael Kaminski and Daniel Zeitlin. Extending finite-memory automata with non-deterministic reassignment (extended abstract). In *AFL*, pages 195–207, 2008.

16 Daniel Kirsten. Distance desert automata and the star height problem. *RAIRO*, 3(39):455–509, 2005.

17 Christof Löding. Logic and automata over infinite trees. Habilitation thesis, Aachen, 2009.

18 Christof Löding. Finding determinisitic subautomata in polynomial time. Personal communication, 2011.

19 D. A. Martin. Borel determinacy. *Ann. Math.*, 102:363–371, 1975.

20 Robert McNaughton. Testing and generating infinite sequences by a finite automaton. *Information and Control*, 9:521–530, 1966.

21 Robert McNaughton and Seymour Papert. *Counter-free Automata*. MIT Press, 1971.

22 David Muller. Infinite sequences and finite machines. In *Switching Theory and Logical Design, Proc. Fourth Annual Symp. IEEE*, pages 3–16. IEEE, 1963.

23 J. Myhill. Finite automata and representation of events. *Fund. Concepts in the Theory of Systems*, pages 57–624, 1957.

24 Sebastian Preugschat and Thomas Wilke. Characterizing fragments of ltl using cuba's. In *FoSSaCS*, 2012.

25 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. soc.*, 141:1–35, 1969.

26 Michael O. Rabin and Dana Scott. Finite automata and their decision problems. *IBM J. Res. and Develop.*, 3:114–125, April 1959.

27 Marcel-Paul Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8:190–194, 1965.

28 Richard Edwin Stearns and Harry B. Hunt III. On the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. *SIAM J. Comput.*, 14(3):598–611, 1985.

29 Larry J. Stockmeyer and Albert R. Meyer. Word problems requiring exponential time: Preliminary report. In *STOC*, pages 1–9, 1973.

Iterative Methods in Combinatorial Optimization

R. Ravi[1]

1 **Tepper School of Business, Carnegie Mellon University**
ravi@cmu.edu

Abstract

In these lectures, I will describe a simple iterative method that supplies new proofs of integrality of linear characterizations of various basic problems in combinatorial optimization, and also allows adaptations to design approximation algorithms for NP-hard variants of these problems involving extra "degree-like" budget constraints. It is inspired by Jain's iterative rounding method for designing approximation algorithms for survivable network design problems, and augmented with a relaxation idea in the work of Lau, Naor, Salvatipour and Singh in their work on designing the approximation algorithm for its degree bounded version. Its application was further refined in recent work of Bansal, Khandekar and Nagarajan on degree-bounded directed network design.

I will begin by reviewing the background material on LP relaxations and their solvability and properties of extreme point or vertex solutions to such problems. I will then introduce the basic framework of the method using the assignment problem, and show its application by re-deriving the approximation results of Shmoys and Tardos for the generalized assignment problem.

I will then discuss linear characterizations for the spanning tree polyhedron in undirected graphs and give a new proof of integrality using an iterative method. I will then illustrate an application to approximating the degree-bounded version of the undirected problem, by proving the results of Goemans and Lau & Singh.

I will continue with showing how these methods for spanning trees simplify and generalize to showing linear descriptions of maximum weight matroid bases and also maximum weight sets that are independent in two different matroids. This also leads to good additive approximation algorithms for a bounded degree version of the matroid basis problem.

I will close with applications of the iterative method by revisiting Jain's original proof for the SNDP and giving a new proof that unifies its treatment with that for the Symmetric TSP polyhedron (describing joint work with Nagarajan and Singh). I will also outline the versatility of the method by pointing out the other problems for which the method has been applied, summarizing the discussion in a recent monograph I have co-authored on this topic with Lap Chi Lau and Mohit Singh (published by Cambridge University Press, 2011).

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

Keywords and phrases combinatorial optimization, linear programming, matroid

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.24

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 24–24

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

On Randomness in Hash Functions*

Martin Dietzfelbinger[1]

1 **Technische Universität Ilmenau**
Ilmenau, Germany
martin.dietzfelbinger@tu-ilmenau.de

Abstract

In the talk, we shall discuss quality measures for hash functions used in data structures and algorithms, and survey positive and negative results. (This talk is *not* about cryptographic hash functions.) For the analysis of algorithms involving hash functions, it is often convenient to assume the hash functions used behave fully randomly; in some cases there is no analysis known that avoids this assumption. In practice, one needs to get by with weaker hash functions that can be generated by randomized algorithms. A well-studied range of applications concern realizations of dynamic dictionaries (linear probing [37], chained hashing, dynamic perfect hashing [21], cuckoo hashing and its generalizations [30, 42]) or Bloom filters [8, 11] and their variants.

A particularly successful and useful means of classification are Carter and Wegman's *universal* or k-wise independent classes, introduced in 1977 [13, 53]. A natural and widely used approach to analyzing an algorithm involving hash functions is to show that it works if a sufficiently strong universal class of hash functions is used [10, 18, 21, 41], and to substitute one of the known constructions of such classes [2, 3, 7, 13, 16, 20, 45, 50, 51, 53]. This invites research into the question of just how much independence in the hash functions is necessary for an algorithm to work. Some recent analyses that gave impossibility results constructed rather artificial classes that would not work [15, 43]; other results pointed out natural, widely used hash classes that would not work in a particular application [1, 25, 26, 41, 43]. Only recently it was shown that under certain assumptions on some entropy present in the set of keys even 2-wise independent hash classes will lead to strong randomness properties in the hash values [14, 39]. The negative results in [25] show that these results may not be taken as justification for using weak hash classes indiscriminately, in particular for key sets with structure.

When stronger independence properties are needed for a theoretical analysis, one may resort to classic constructions [46, 47, 22]. Only in 2003 it was found out how full randomness can be *simulated* using only linear space overhead (which is optimal) [28, 40]. The "split-and-share" approach [17, 24, 30] can be used to justify the full randomness assumption in some situations in which full randomness is needed for the analysis to go through, like in many applications involving multiple hash functions (e. g., generalized versions of cuckoo hashing with multiple hash functions [19, 30, 31, 32, 33, 34, 38] or larger bucket sizes [27, 29, 12], load balancing [5], Bloom filters and variants [8, 23], or minimal perfect hash function constructions [6, 9, 17]).

For practice, efficiency considerations beyond constant factors are important. It is not hard to construct very efficient 2-wise independent classes [16, 48]. Using k-wise independent classes for constant k bigger than 3 has become feasible in practice only by new constructions [36, 50, 51] involving tabulation. This goes together well with the quite new result that linear probing works with 5-independent hash functions [41].

Recent developments suggest that the classification of hash function constructions by their degree of independence alone may not be adequate in some cases. Thus, one may want to analyze the behaviour of specific hash classes in specific applications, circumventing the concept of k-wise independence. Several such results were recently achieved concerning hash functions that utilize tabulation [44, 49]. In particular if the analysis of the application involves using

* This work was partially supported by DFG grant DI 412/10-2.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 25–28

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

randomness properties in graphs and hypergraphs (generalized cuckoo hashing [30], also in the version with a "stash" [35], or load balancing [5, 52]), a hash class combining k-wise independence with tabulation has turned out to be very powerful [4, 28, 54].

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems–Computations on discrete structures–Sorting and searching, E.2 Data Storage Representations–Hash-table representations

Keywords and phrases Algorithms, hash functions, randomized algorithms, data structures, graphs, hypergraphs

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.25

References

1 N. Alon, M. Dietzfelbinger, P. B. Miltersen, E. Petrank, and G. Tardos. Linear hash functions. *J. ACM*, 46(5):667–683, 1999.
2 N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Simple construction of almost k-wise independent random variables. *Random Struct. Algorithms*, 3(3):289–304, 1992.
3 N. Alon, O. Goldreich, J. Håstad, and R. Peralta. Addendum to "Simple construction of almost k-wise independent random variables". *Random Struct. Algorithms*, 4(1):119–120, 1993.
4 M. Aumüller. An alternative analysis of cuckoo hashing with a stash and realistic hash functions. Diplomarbeit, Technische Universität Ilmenau, March 2010.
5 Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal. Balanced allocations. *SIAM J. Comput.*, 29(1):180–200, 1999.
6 D. Belazzougui, F. C. Botelho, and M. Dietzfelbinger. Hash, displace, and compress. In *Proc. 17th ESA*, LNCS 5757, pages 682–693. Springer, 2009.
7 C. Bertram-Kretzberg and H. Lefmann. mod_p-Tests, almost independence and small probability spaces. *Random Struct. Algorithms*, 16(4):293–313, 2000.
8 B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, 1970.
9 F. C. Botelho, R. Pagh, and N. Ziviani. Simple and space-efficient minimal perfect hash functions. In *Proc. 10th WADS*, LNCS 4619, pages 139–150. Springer, 2007.
10 A. Z. Broder and A. R. Karlin. Multilevel adaptive hashing. In *Proc. 1st ACM-SIAM SODA*, pages 43–53, 1990.
11 A. Z. Broder and M. Mitzenmacher. Survey: Network applications of bloom filters: A survey. *Internet Mathematics*, 1(4), 2003.
12 J. A. Cain, P. Sanders, and N. C. Wormald. The random graph threshold for k-orientiability and a fast algorithm for optimal multiple-choice allocation. In *Proc. 18th ACM-SIAM SODA*, pages 469–476. SIAM, 2007.
13 L. Carter and M. N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2):143–154, 1979.
14 K.-M. Chung and S. P. Vadhan. Tight bounds for hashing block sources. In *Proc. 11th APPROX/12th RANDOM*, LNCS 5171, pages 357–370. Springer, 2008.
15 J. Cohen and D. M. Kane. Bounds on the independence required for cuckoo hashing. Manuscript, 2009. `http://math.stanford.edu/~dankane/cuchkoohashing.pdf` (last download: 2012-01-02).
16 M. Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *Proc. 13th STACS*, LNCS 1046, pages 569–580. Springer, 1996.

17 M. Dietzfelbinger. Design strategies for minimal perfect hash functions. In *Proc. 4th SAGA*, LNCS 4665, pages 2–17. Springer, 2007.

18 M. Dietzfelbinger, J. Gil, Y. Matias, and N. Pippenger. Polynomial hash functions are reliable (extended abstract). In *Proc. 19th ICALP*, LNCS 443, pages 235–246. Springer, 1992.

19 M. Dietzfelbinger, A. Goerdt, M. Mitzenmacher, A. Montanari, R. Pagh, and M. Rink. Tight thresholds for cuckoo hashing via XORSAT. In *Proc. 37th ICALP, Part I*, LNCS 6198, pages 213–225, 2010.

20 M. Dietzfelbinger, T. Hagerup, J. Katajainen, and M. Penttonen. A reliable randomized algorithm for the closest-pair problem. *J. Algorithms*, 25(1):19–51, 1997.

21 M. Dietzfelbinger, A. R. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan. Dynamic perfect hashing: Upper and lower bounds. *SIAM J. Comput.*, 23(4):738–761, 1994.

22 M. Dietzfelbinger and F. Meyer auf der Heide. A new universal class of hash functions and dynamic hashing in real time. In *Proc. 17th ICALP*, LNCS 443, pages 6–19. Springer, 1990.

23 M. Dietzfelbinger and R. Pagh. Succinct data structures for retrieval and approximate membership (extended abstract). In *Proc. 35th ICALP, Part I*, LNCS 5125, pages 385–396. Springer, 2008.

24 M. Dietzfelbinger and M. Rink. Applications of a splitting trick. In *Proc. 36th ICALP, Part I*, LNCS 5555, pages 354–365. Springer, 2009.

25 M. Dietzfelbinger and U. Schellbach. On risks of using cuckoo hashing with simple universal hash classes. In *Proc. 20th ACM-SIAM SODA*, pages 795–804, 2009.

26 M. Dietzfelbinger and U. Schellbach. Weaknesses of cuckoo hashing with a simple universal hash class: The case of large universes. In *Proc. 35th SOFSEM*, LNCS 5404, pages 217–228. Springer, 2009.

27 M. Dietzfelbinger and C. Weidling. Balanced allocation and dictionaries with tightly packed constant size bins. *Theor. Comput. Sci.*, 380(1-2):47–68, 2007.

28 M. Dietzfelbinger and P. Woelfel. Almost random graphs with simple hash functions. In *Proc. 35th ACM STOC*, pages 629–638, 2003.

29 D. Fernholz and V. Ramachandran. The k-orientability thresholds for $g_{n,p}$. In *Proc. 18th ACM-SIAM SODA*, pages 459–468. SIAM, 2007.

30 D. Fotakis, R. Pagh, P. Sanders, and P. G. Spirakis. Space efficient hash tables with worst case constant access time. *Theory Comput. Syst.*, 38(2):229–248, 2005.

31 N. Fountoulakis, M. Khosla, and K. Panagioutou. The multiple-orientability thresholds for random hypergraphs. In *Proc. 22nd ACM-SIAM SODA*, pages 1222–1236. SIAM, 2011.

32 N. Fountoulakis and K. Panagiotou. Orientability of random hypergraphs and the power of multiple choices. In *Proc. 37th ICALP, Part I*, LNCS 6198, pages 348–359. Springer, 2010.

33 A. M. Frieze and P. Melsted. Maximum matchings in random bipartite graphs and the space utilization of cuckoo hashtables. CoRR, arXiv:0910.5535, 2009.

34 P. Gao and N. C. Wormald. Load balancing and orientability thresholds for random hypergraphs. In *Proc. 42nd ACM STOC*, pages 97–104. ACM, 2010.

35 A. Kirsch, M. Mitzenmacher, and U. Wieder. More robust hashing: Cuckoo hashing with a stash. In *Proc. 16th ESA*, LNCS 5193, pages 611–622. Springer, 2008.

36 T. Q. Klassen and P. Woelfel. Independence of tabulation-based hash classes. CoRR, arXiv:1112.3323, 2011.

37 D. E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching.* Addison-Wesley, 1973.

38 E. Lehman and R. Panigrahy. 3.5-way cuckoo hashing for the price of 2-and-a-bit. In *Proc. 17th ESA*, LNCS 5757, pages 671–681. Springer, Springer, 2009.

39 M. Mitzenmacher and S. P. Vadhan. Why simple hash functions work: Exploiting the entropy in a data stream. In *Proc. 19th ACM-SIAM SODA*, pages 746–755, 2008.

40 A. Pagh and R. Pagh. Uniform hashing in constant time and optimal space. *SIAM J. Comput.*, 38(1):85–96, 2008.

41 A. Pagh, R. Pagh, and M. Ružić. Linear probing with constant independence. *SIAM J. Comput.*, 39(3):1107–1120, 2009.

42 R. Pagh and F. F. Rodler. Cuckoo hashing. *J. Algorithms*, 51(2):122–144, 2004.

43 M. Pătraşcu and M. Thorup. On the k-independence required by linear probing and minwise independence. In *Proc. 37th ICALP, Part I*, LNCS 6198, pages 715–726, 2010.

44 M. Pătraşcu and M. Thorup. The power of simple tabulation hashing. In *Proc. 43rd ACM STOC*, pages 1–10, 2011.

45 E. Porat and B. Shalem. Another one flew over the cuckoo's nest. CoRR, arXiv:1104.5400, 2011.

46 A. Siegel. On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications (extended abstract). In *30th IEEE FOCS*, pages 20–25, 1989.

47 A. Siegel. On universal classes of extremely random constant-time hash functions. *SIAM J. Comput.*, 33(3):505–543, 2004.

48 M. Thorup. Even strongly universal hashing is pretty fast. In *Proc. 11th ACM-SIAM SODA*, pages 496–497, 2000.

49 M. Thorup. String hashing for linear probing. In *Proc. 20th ACM-SIAM SODA*, pages 655–664, 2009.

50 M. Thorup and Y. Zhang. Tabulation based 4-universal hashing with applications to second moment estimation. In *Proc. 15th ACM-SIAM SODA*, pages 615–624, 2004.

51 M. Thorup and Y. Zhang. Tabulation based 5-universal hashing and linear probing. In *Proc. 12th ALENEX 2010*, pages 62–76. SIAM, 2010.

52 B. Vöcking. How asymmetry helps load balancing. *J. ACM*, 50(4):568–589, 2003.

53 M. N. Wegman and L. Carter. New hash functions and their use in authentication and set equality. *J. Comput. Syst. Sci.*, 22(3):265–279, 1981.

54 P. Woelfel. Asymmetric balanced allocation with simple hash functions. In *Proc. 17th ACM-SIAM SODA*, pages 424–433, 2006.

Pseudo-deterministic Algorithms*

Shafi Goldwasser[1]

1 **MIT and Weizmann Institute of Science**
Cambridge, MA, USA and Rehovot, Israel
shafi@csail.mit.edu

Abstract

In this talk we describe a new type of probabilistic algorithm which we call *Bellagio* Algorithms: a randomized algorithm which is guaranteed to run in expected polynomial time, and to produce a correct and unique solution with high probability. These algorithms are pseudo-deterministic: they can not be distinguished from deterministic algorithms in polynomial time by a probabilistic polynomial time observer with black box access to the algorithm.

We show a necessary and sufficient condition for the existence of a Bellagio Algorithm for an NP relation R: R has a Bellagio algorithm if and only if it is deterministically reducible to some decision problem in BPP. Several examples of Bellagio algorithms, for well known problems in algebra and graph theory which improve on deterministic solutions, follow.

The notion of pseudo-deterministic algorithms (or more generally computations) is interesting beyond just sequential algorithms. In particular, it has long been known that it is impossible to solve deterministically tasks such as *consensus* in a faulty distributed systems, whereas randomized protocols can achieve consensus in expected constant time. We thus explore the notion of pseudo-deterministic fault tolerant distributed protocols: randomized protocols which are polynomial time indistinguishable from deterministic protocols in presence of faults.

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

Keywords and phrases randomized algorithms, distributed computing, Monte Carlo, Las Vegas

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.29

* This work was partially supported by NSF Contract CCF-0729011 and ISF grant 700/08

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 29–29
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$\frac{13}{9}$-approximation for Graphic TSP*

Marcin Mucha[1]

1 **Institute of Informatics, University of Warsaw,** mucha@mimuw.edu.pl

Abstract

The Travelling Salesman Problem is one of the most fundamental and most studied problems in approximation algorithms. For more than 30 years, the best algorithm known for general metrics has been Christofides's algorithm with approximation factor of $\frac{3}{2}$, even though the so-called Held-Karp LP relaxation of the problem is conjectured to have the integrality gap of only $\frac{4}{3}$. Very recently, significant progress has been made for the important special case of graphic metrics, first by Oveis Gharan et al. [3], and then by Mömke and Svensson [8]. In this paper, we provide an improved analysis of the approach presented in [8] yielding a bound of $\frac{13}{9}$ on the approximation factor, as well as a bound of $\frac{19}{12}+\varepsilon$ for any $\varepsilon > 0$ for a more general Travelling Salesman Path Problem in graphic metrics.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases approximation algorithms, travelling salesman problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.30

1 Introduction and related work

The Travelling Salesman Problem (TSP) is one the most fundamental and most studied problems in combinatorial optimization, and aproximation algorithms in particular. In the most standard version of the problem, we are given a metric (V, d) and the goal is to find a closed tour that visits each point of V exactly once and has minimum total cost, as measured by d. This problem is APX-hard, and the best known approximation factor of $\frac{3}{2}$ was obtained by Christofides [1] more than thirty years ago. However, the so-called Held-Karp LP relaxation of TSP is conjectured to have an integrality gap of $\frac{4}{3}$. It is known to have a gap at least that big, however the best known upper bound [9] for the gap is given by Christofides's algorithm and equal to $\frac{3}{2}$.

In a more general version of the problem, called the Travelling Salesman Path Problem (TSPP), in addition to a metric (V, d) we are also given two points $s, t \in V$ and the goal is to find a path from s to t visiting each point exactly once, except if s and t are the same point in which case it can be visited twice (this is when TSPP reduces to TSP). For this problem, the best approximation algorithm known is that of Hoogeveen [7] with approximation factor of $\frac{5}{3}$. However, the Held-Karp relaxation of TSPP is conjectured to have an integrality gap of $\frac{3}{2}$.

One of the natural directions of attacking these problems is to consider special cases and several attempts of this nature has been made. The most interesting one is the graphic TSP/TSPP, where we assume that the given metric is the shortest path metric of an undirected graph. Equivalently, in graphic TSP we are given an undirected graph $G = (V, E)$ and we need to find a shortest tour that visits each vertex *at least once*. Yet another formulation

* This work was partially supported by the Polish Ministry of Science grant N206 355636 and by the ERC StD project PAAl no. 259515

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 30–41

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

would ask for a minimum size Eulerian multigraph spanning V and only using edges of G. Similar formulations apply to the graphic TSPP case. The reason why these special cases are very interesting is that they seem to include the difficult inputs of TSP/TSPP. Not only are they APX-hard (see [5]), but also the standard examples showing that the Held-Karp relaxation has a gap of at least $\frac{4}{3}$ in the TSP case and $\frac{3}{2}$ in the TSPP case, are in fact graphic metrics.

Very recently, significant progress has been made in approximating the graphic TSP and TSPP. First, Oveis Gharan et al. [3] gave an algorithm with an approximation factor $\frac{3}{2}-\varepsilon$ for graphic TSP. Following that, Mömke and Svensson [8] obtained a significantly better approximation factor of $\frac{14(\sqrt{2}-1)}{12\sqrt{2}-13} \approx 1.461$ for graphic TSP, as well as factor $3-\sqrt{2}+\varepsilon \approx 1.586+\varepsilon$ for graphic TSPP, for any $\varepsilon > 0$. Their approach uses matchings in a truly ingenious way. Whereas most earlier approaches (including that of Christofides [1] as well as Oveis Gharan et al. [3]) add edges of a matching to a spanning tree to make it Eulerian, the new approach is based on adding and removing the matching edges. This process is guided by a so-called removable pairing of edges which essentially encodes the information on which edges can be simultanously removed from the graph without disconnecting it. A large removable pairing of edges is found by computing a minimum cost circulation in a certain auxiliary flow network, and the bounds on the cost of this circulation translate into bounds on the size of the resulting TSP tour/path.

1.1 Our results

In this paper we present an improved analysis of the cost of the circulation used in the construction of the TSP tour/path. Our results imply a bound of $\frac{13}{9} \approx 1.444$ on the approximation factor for the graphic TSP, as well as a $\frac{19}{12}+\varepsilon \approx 1.583+\varepsilon$ bound for the graphic TSPP, for any $\varepsilon > 0$. The circulation used in [8] consists of two parts: the "core" part based on an extreme optimal solution to the Held-Karp relaxation of TSP, and the "correction" part that adds enough flow to the core part to make it feasible. We improve bounds on costs of both part, in particular we show that the second part is in a sense free. As for the first part, similarly to the original proof of Mömke and Svensson, our proof exploits its knapsack-like structure. However, we use the 2-dimensional knapsack problem in our analysis, instead of the standard knapsack problem. We also provide a supplementary essentially matching lower bound on the cost of the core part, which means that any further progress on bounding that cost has to take into account more than just the knapsack-like structure of the circulation.

1.2 Organization of the paper

In the next section we present previous results relevant to the contributions of this paper. In particular we recall key definitions and theorems of Mömke and Svensson [8]. In Section 3 we present the improved upper bound on the cost of the core part of the circulation, as well as an essentially matching lower bound. In Section 4 we prove that the correction part of the circulation is in a sense free. Finally, in Section 5 we apply the results of the previous sections to obtain improved approximation algorithms for graphic TSP and TSPP.

2 Preliminaries

In this section we review some standard results concerning TSP/TSPP approximation and recall the parts of the work of Mömke and Svensson [8] relevant to the contributions of

this paper. Note that large parts of the material presented in [8] are omitted entirely or collapsed to a single theorem statement. A reader interested in a more detailed and complete exposition is advised to read the original paper instead.

2.1 Held-Karp Relaxation and the Algorithm of Christofides

The Held-Karp relaxation (or subtour elimination LP) for graphic TSP on graph $G = (V, E)$ can be formulated as follows (see [6, 4, 8] for details on equivalence between different formulations):

$$\min \sum_{e \in E} x_e \text{ subject to } x(\delta(S)) \geq 2 \text{ for } \emptyset \neq S \subset V, \text{ where } x_e \geq 0.$$

Here $\delta(S)$ denotes the set of all edges between S and $V \setminus S$ for any $S \subseteq V$, and $x(F)$ denotes $\sum_{e \in F} x_e$ for any $F \subseteq E$. We will refer to this LP as $LP(G)$ and denote the value of any of its optimal solutions by $\mathrm{OPT}_{LP}(G)$.

The approximation ratio of the classic $\frac{3}{2}$-approximation algorithm for metric TSP due to Christofides [1] is in fact related to $\mathrm{OPT}_{LP}(G)$ as follows:

▶ **Theorem 2.1** (Shmoys, Williamson [9]). *The cost of the solution produced by the algorithm of Christofides on a graph G is bounded by $n + OPT_{LP}(G)/2$, and so its approximation factor is at most $\frac{n+OPT_{LP}(G)/2}{OPT_{LP}(G)}$.*

The Held-Karp relaxation can be generalized to the graphic TSPP in a straightforward manner. Suppose we want to solve the problem for a graph $G = (V, E)$ and endpoints s, t. Let $\Phi = \{S \subseteq V : |\{s, t\} \cap S| = 1\}$. Then the relaxation can be written as

$$\begin{array}{lrcll} \min & \sum_{e \in E} x_e & & & \\ \text{subject to} & x(\delta(S)) & \geq & 2 & \text{for } S \notin \Phi \\ & x(\delta(S)) & \geq & 1 & \text{for } S \in \Phi \\ & x_e & \geq & 0 & \text{for } e \in E \end{array}$$

We denote this generalized program by $LP(G, s, t)$ and its optimum value by $\mathrm{OPT}_{LP}(G, s, t)$. It is clear that $\mathrm{OPT}_{LP}(G, v, v) = \mathrm{OPT}_{LP}(G)$ for any $v \in V$.

Let $G' = (V, E \cup \{e'\})$, where $e' = \{s, t\}$. From any feasible solution to $LP(G, s, t)$ we can obtain a feasible solution to $LP(G')$ by adding 1 to $x_{e'}$. Therefore

▶ **Fact 2.2.** *$OPT_{LP}(G, s, t) \geq OPT_{LP}(G') - 1$.*

2.2 Reduction to Minimum Cost Circulation

The authors of [8] use the optimal solution of $LP(G)$ to construct a low cost circulation in a certain auxiliary flow network. This circulation is then used to produce a small TSP tour for G. We will now describe the construction of the flow network and the relationship between the cost of the circulation and the size of the TSP tour.

Let us start with the following reduction

▶ **Lemma 2.3** (Lemma 2.1 and Lemma 2.1(generalized) of Mömke and Svensson [8]). *If there exists a polynomial time algorithm that for any 2-vertex connected graph G returns a graphic TSP solution of cost at most $r \cdot OPT_{LP}(G)$, then there exists an algorithm that does the same for any connected graph. Similarly, if there exists a polynomial time algorithm that for any 2-vertex connected graph G and its two vertices s, t returns a graphic TSPP solution of cost at most $r \cdot OPT_{LP}(G, s, t)$, then there exists an algorithm that does the same for any connected graph.*

We will henceforth assume that the graphs we work with are all 2-vertex-connected. Let G be such graph. We now construct a certain auxiliary flow network corresponding to G.

Let T be a DFS spanning tree of G with an arbitrary root vertex r. Direct all edges of T (called *tree-edges*) away from the root, and all other edges (called *back-edges*) towards the root. Let $\vec{G}$ be the resulting directed graph, and let $\vec{T}$ be its subgraph corresponding to T. Where neccessary to avoid confusion, we will use the name *arcs* (and *tree-arcs* and *back-arcs*) for the edges of this directed graph. The flow network is obtained from $\vec{G}$ by replacing some of its vertices with gadgets, as described below.

Let v be any non-root vertex of $\vec{G}$ having l children: $w_1, \ldots, w_l$ in T. We introduce l new vertices $v_1, \ldots, v_l$ and replace the tree-arc (v, w_j) by tree-arcs (v, v_j) and (v_j, w_j) for $j = 1, \ldots, l$. We also redirect to v_j all the back-arcs leaving the subtree rooted by w_j and entering v. We will call the new vertices and the root *in-vertices* and the remaining vertices *out-vertices*. We will also denote the set of all in-vertices by $\mathcal{I}$. Notice that all the back-arcs go from out-vertices to in-vertices, and that each in-vertex has exactly one outgoing edge.

We assign lower bounds (demands) and upper bounds (capacities) as well as costs to arcs. The demands of the tree-arcs are 1 and the demands of the back-arcs are 0. The capacities of all arcs are ∞. Finally the cost of any circulation f is defined to be $\sum_{v \in \mathcal{I}} \max(f(B(v)) - 1, 0)$, where $B(v)$ is the set of incoming arcs of v. This basically means that the cost is 0 for tree-arcs and 1 for back-arcs, except that for every in-vertex the first unit of circulation is free. The circulation network described above will be denoted $C(G, T)$. For any circulation C, we will use $|C|$ to denote its cost as described above.

It is worth noting that the cost function of $C(G, T)$ can be simulated using the usual fixed-cost edges by introducing an extra vertex v' for each in-vertex v, redirecting all in-arcs of v to v' and putting two arcs from v' to v: one with capacity of 1 and cost 0, and the other with capacity ∞ and cost 1. For simplicity of presentation however, we will use the simpler network with a slighly unusual cost function.

Also note that the edges of $C(G, T)$ minus the incoming tree edges of the in-vertices are in 1-to-1 correspondance with the edges of G. Similarly, all vertices of $C(G, T)$ except for the new in-vertices correspond to the vertices of the original graph. We will often use the same symbol to denote both edges or both vertices.

The main technical tool of [8] is given by the following theorem:

▶ **Theorem 2.4** (Lemma 4.1 of [8]). *Let G be a 2-vertex connected graph, let T be a DFS tree of G, and let C^* be a circulation in $C(G, T)$ of cost $|C^*|$. Then there exists a spanning Eulerian multigraph H in G with at most $\frac{4}{3}n + \frac{2}{3}|C^*| - \frac{2}{3}$ edges. In particular, this means that there exists a TSP tour in the shortest path metric of G with the same cost.*

and its generalized version

▶ **Theorem 2.5** (Lemma 4.1(generalized) of [8]). *Let $G = (V, E)$ be a 2-vertex connected graph and s, t its two vertices, and let $G' = (V, E \cup \{e'\})$ where $e' = \{s, t\}$. Let T be a DFS tree of G' and let C^* be a circulation in $C(G', T)$ of cost $|C^*|$. Then there exists a spanning multigraph H in G, that has an Eulerian path between s and t with at most $\frac{4}{3}n + \frac{2}{3}|C^*| - \frac{2}{3} + dist_G(s, t)$ edges. In particular, this means that there exists a TSP path between s and t in the shortest path metric of G with the same cost.*

▶ Remark. The above theorem is not just a rewording of the generalized version of Lemma 4.1 from [8]. In our version C^* is a circulation in $C(G', T)$ and not $C(G, T)$. Note however, that in the proof of Theorem 1.2 of [8] the authors are in fact using the version above, and provide arguments for why it is correct.

In order to be able to apply Theorem 2.4 and Theorem 2.5, the authors of [8] use the optimal solution of $LP(G)$ to define a circulation f in $C(G,T)$ as follows. Let $G=(V,E)$ be a graph, and let $E'=\{e\in E : x^*_e>0\}$, where x^* is an extreme optimal solution of $LP(G)$. Let $G'=(V,E')$. It is clear that x^* is also an optimal solution for $LP(G')$, so an r-approximate TSP tour with respect to $\mathrm{OPT}_{LP}(G')$ is also r-approximate with respect to $\mathrm{OPT}_{LP}(G)$. Therefore, we can always assume that $E'=E$. The reason why this assumption is useful is given by the following theorem.

▶ **Theorem 2.6** (Cornuejols, Fonlupt, Naddef [2]). *For any graph G, the support of any extreme optimal solution to $LP(G)$ has size at most $2n-1$.*

Thus, we can assume that $|E|\leq 2n-1$. Moreover, we can assume that G is 2-vertex connected because of Lemma 2.3.

Let T used in the construction of $C(G,T)$ be the tree resulting from always following the edge e with the highest value of x^*_e. We construct a circulation f in $C(G,T)$ as a sum of two ciculations: f' and f''. The ciculation f' corresponds to sending, for each back-arc a, flow of size $\min(x^*_a,1)$ along the unique cycle formed by a and some tree-arcs. The circulation f'' is defined in a way that guarantees that $f=f'+f''$ satisfies all the lower bounds. Let v be an out-vertex and w an in-vertex, such that there is an arc (v,w) in $C(G,T)$, and the flow on (v,w) is smaller than 1. Also let a be any back-arc going from a descendant of w to an ancestor of v (in $\vec{T}$). Such an arc always exists since G is 2-vertex connected. We push flow along all edges of the unique cycle formed by a and tree-arcs until the flow on (v,w) reaches 1.

The total cost of f can be bounded by

$$\sum_{v\in\mathcal{I}}\max(f(B(v))-1,0)\leq\sum_{v\in\mathcal{I}}\max(f'(B(v))-1,0)+\sum_{v\in\mathcal{I}}f''(B(v)).$$

We will denote the sum $\sum_{v\in\mathcal{I}}f''(B(v))$ by $|f''|$, which is slightly inconsistent with previous definitions, but simplifies the notation quite a bit. We thus have $|f|\leq|f'|+|f''|$.

The authors of [8] provide the following bounds for the two terms of the above expression:

▶ **Lemma 2.7** (Claim 5.3 in [8]). $|f''|\leq OPT_{LP}(G)-n$.

▶ **Lemma 2.8** (Claim 5.4 in [8]). $|f'|\leq(7-6\sqrt{2})n+4(\sqrt{2}-1)OPT_{LP}(G)$.

The main theorem of [8] follows from these two bounds

▶ **Theorem 2.9** (Theorem 1.1 in [8]). *There exists a polynomial time approximation algorithm for graphic TSP with approximation ratio $\frac{14(\sqrt{2}-1)}{12\sqrt{2}-13}<1.461$.*

3 New upper bound for $|f'|$

In this section we describe an improved bound on $|f'|$.

▶ **Lemma 3.1.** $|f'|\leq\frac{5}{3}OPT_{LP}-\frac{3}{2}n$.

Before presenting our analysis of the cost of f' let us recall some notation and basic observations introduced in [8]. For any $v\in\mathcal{I}$ let t_v be the (unique) outgoing arc of v.

▶ **Fact 3.2.** *For every in-vertex v, we have $|B(v)|\geq\left\lceil\frac{f'(B(v))}{\min(x^*_{t_v},1)}\right\rceil$.*

Proof. Since T was constructed by always following the arc a with the highest value of x^*_a, we have that $x^*_{t_v}\geq x_a$ for any $a\in B(v)$ and the claim follows. ◀

Decompose $f'(B(v))$ into two parts: $l_v = \min(2 - x^*_{t_v}, f'(B(v)))$ and $u_v = f'(B(v)) - l_v$. The intuition here is that the higher u_v is, the larger $\mathrm{OPT}_{LP}(G)$ is. In particular, if we let $u^* = \sum_{v\in\mathcal{I}} u_v$, then

▶ **Fact 3.3** (Stated in the proof of Claim 5.4 in [8]). $u^* \leq 2(OPT_{LP}(G) - n)$.

Proof. Consider a vertex v of G which (in the construction of $C(G,T)$) is replaced by a gadget with a set $\mathcal{I}_v$ of in-vertices, and let $x^*(v)$ be the fractional degree of v in x^*. Since for any $w \in \mathcal{I}_v$, the tree-arc t_w and all the back-arcs entering w correspond to edges of G incident to v, each such w contributes at least $2 + u_w$ to $x^*(v)$, provided that $u_w > 0$ (if $u_w = 0$ we cannot bound w's contribution in any way). Since we also know that $x^*(v) \geq 2$ (this is one of the inequalities of the Held-Karp relaxation), we get the following bound

$$x^*(v) \geq \max\left(2, \sum_{w\in\mathcal{I}_v,\, u_w>0} (2+u_w)\right) \geq 2 + \sum_{w\in\mathcal{I}_v} u_w.$$

Summing this over all vertices we get $2\mathrm{OPT}_{LP}(G) \geq 2n + u^*$, and the claim follows. ◀

Because of Theorem 2.6 we have $\sum_{v\in\mathcal{I}} |B(v)| + n - 1 \leq 2n - 1$, and so by Fact 3.2

$$\sum_{v\in\mathcal{I}} \left\lceil \frac{l_v + u_v}{\min(1, x^*_{t_v})} \right\rceil \leq n.$$

Also note that in terms of l_v and u_v the total cost of f' is given by the following formula

$$\sum_{v\in\mathcal{I}} \max(0, l_v + u_v - 1).$$

Our goal is to upper-bound this cost as a function of n and u^*. Instead of working directly with G and the solution x^* to the corresponding $LP(G)$, we abstract out the key properties of $x^*_{t_v}$, l_v and u_v and work in this restricted setting.

▶ **Definition 3.4.** A *configuration* of size n is a triple (x, l, u), where $x, l, u : \{1, \ldots, n\} \to \mathbb{R}_{\geq 0}$ such that for all $i = 1, \ldots, n$

1. $0 < x_i \leq 1$,
2. $l_i \leq 2 - x_i$, and
3. $u_i > 0 \implies l_i = 2 - x_i$.

▶ **Definition 3.5.** Let $C = (x, l, u)$ be a configuration. We will say that the i-th element of C uses $\lceil \frac{l_i+u_i}{x_i} \rceil$ *edges* and denote this number by $e_i(C)$, or e_i if it is clear what C is. We will also say that C uses $\sum_{i=1}^n e_i$ edges. Also, the *value* of C is defined as $\mathrm{val}(C) = \sum_{i=1}^n \max(0, l_i + u_i - 1)$. Note that upper-bounding the cost of f' corresponds to finding the maximum value of C.

▶ Remark. The values x_i, l_i and u_i correspond to x_{t_v}, l_v and u_v, respectively. The properties enforced on the former are clearly satisfied by the latter with the exception of the inequalities $x_i \leq 1$. The reason for introducing these inequalities is the following. Without them, the natural definition of the number of edges used by the i-th element of C would be $\left\lceil \frac{l_i+u_i}{\min(x_i,1)} \right\rceil$. However, in that case, for any configuration C there would exists a configuration C' with $\mathrm{val}(C') \leq \mathrm{val}(C)$ and $x_i \leq 1$ for all $i = 1, \ldots, n$. In order to construct C' simply replace all $x_i > 1$ with ones. If as a result we get $l_i < 2 - x_i$ and $u_i > 0$ for some i, simultanously decrease u_i and increase l_i at the same rate until one of these inequalities becomes an equality.

For that reason, we prefer to simply assume $x_i \leq 1$ and be able to use a (slightly) simpler definition of e_i. As we will see, the inequalities $x_i \leq 1$ turn out to be quite useful as well.

We denote by $\mathrm{CONF}(n, u^*)$ the set of all configurations (x, l, u) of size n such that $\sum_{i=1}^{n} u_i = u^*$. We also use $\mathrm{OPT}(n, u^*)$ to denote any maximum value element of $\mathrm{CONF}(n, u^*)$, and $\mathrm{VAL}(n, u^*)$ to denote its value. It is easy to see that

▶ **Fact 3.6.** $|f'| \leq VAL(n, u^*)$.

Notice that determining $\mathrm{VAL}(n, u^*)$ for given n and u^* is a 2-dimensional knapsack problem. Here, items are the possible triples (x_i, l_i, u_i) satisfying the configuration definition. The value of such a triple is equal to $\max(0, l_i + u_i - 1)$, i.e. its contribution to the configuration value, if used in one. Also, the "mass" of (x_i, l_i, u_i) is u_i and its "volume" is e_i. We want to maximize the total item value, while keeping tht total mass $\leq u^*$ and total volume $\leq n$.

▶ **Lemma 3.7.** *For any $n \in \mathbb{N}, u^* \in \mathbb{R}_{\geq 0}$, there exists an optimal configuration in $CONF(n, u^*)$ such that:*

1. $e_i = \frac{l_i + u_i}{x_i}$ *for all $i = 1, \ldots, n$ (in particular, all e_i are integral),*
2. $(l_i = 0) \vee (l_i = 2 - x_i)$ *for all $i = 1, \ldots, n$.*

Proof. We prove each property by showing a way to transform any $C \in \mathrm{CONF}(n, u^*)$ into $C' \in \mathrm{CONF}(n, u^*)$ such that $\mathrm{val}(C') \geq \mathrm{val}(C)$ and C' satisfies the property.

Let us start with the first property, which basically says that all edges are fully saturated. Assume we have $e_i > \frac{l_i + u_i}{x_i}$ for some $i \in \{1, \ldots, n\}$. If $l_i < 2 - x_i$, we increase l_i until either $e_i = \frac{l_i + u_i}{x_i}$, in which case we are done, or $l_i = 2 - x_i$. In the second case we start decreasing x_i while increasing l_i at the same rate, until $e_i = \frac{l_i + u_i}{x_i}$. Clearly, both transformations increase the value of the configuration and keep both u_i and e_i unchanged.

To prove the second property, let us assume that for some $i \in \{1, \ldots, n\}$ we have $0 < l_i < 2 - x_i$. We also assume that our configuration already satisfies the first property, in particular we have $e_i = \frac{l_i}{x_i}$ ($u_i = 0$ since $l_i < 2 - x_i$). We increase x_i and keep $l_i = e_i x_i$ until $l_i + x_i = 2$. This increases the value of the configuration and keeps u_i and e_i unchanged. To see that $x_i \leq 1$, note that $x_i = l_i / e_i \leq l_i$ and $x_i + l_i = 2$. ◀

▶ **Theorem 3.8.** *For any $n \in \mathbb{N}, u^* \in \mathbb{R}_{\geq 0}$, and any $C \in CONF(n, u^*)$ we have $val(C) \leq u^* + \frac{1}{6}(n - u^*)$.*

Proof. It is enough to prove the bound for optimal configurations satisfying the properties in Lemma 3.7. Let C be such a configuration. We will prove that for all $i = 1, \ldots, n$ we have:

$$\mathrm{val}_i = \max(0, l_i + u_i - 1) \leq u_i + \frac{1}{6}(e_i - u_i).$$

Summing this bound over all i gives the desired claim.

If $u_i = l_i = e_i = 0$, then the bound clearly holds. It follows from Lemma 3.7 that the only other case to consider is when $l_i = 2 - x_i$ and $e_i = \frac{l_i + u_i}{x_i}$ (notice that in this case $l_i + u_i - 1 \geq 0$ and so $\mathrm{val}_i = l_i + u_i - 1$).It follows from these two equalities that $e_i x_i = l_i + u_i = 2 - x_i + u_i$ and so

$$x_i = \frac{2 + u_i}{1 + e_i}.$$

Using this expression to bound val_i we get

$$\mathrm{val}_i = l_i + u_i - 1 = 2 - x_i + u_i - 1 = 1 + u_i - x_i = 1 + u_i - \frac{2 + u_i}{1 + e_i} = u_i - \frac{1 - (e_i - u_i)}{1 + e_i}.$$

We need to prove that

$$u_i - \frac{1 - (e_i - u_i)}{1 + e_i} \leq u_i + \frac{1}{6}(e_i - u_i),$$

or equivalently

$$(e_i - u_i)\left(\frac{1}{6} - \frac{1}{1+e_i}\right) + \frac{1}{1+e_i} \geq 0.$$

Since $u_i \leq e_i$ (this follows from property 1 in Lemma 3.7 and the fact that $x_i \leq 1$), we have two cases to consider.

Case 1: $\frac{1}{6} - \frac{1}{1+e_i} \geq 0$. In this case the whole expression is clearly nonnegative.

Case 2: $\frac{1}{6} - \frac{1}{1+e_i} < 0$, meaning that $e_i \in \{1,2,3,4\}$. In this case we proceed as follows:

$$(e_i - u_i)\left(\frac{1}{6} - \frac{1}{1+e_i}\right) + \frac{1}{1+e_i} = u_i\left(\frac{1}{1+e_i} - \frac{1}{6}\right) + \frac{e_i}{6} - \frac{e_i - 1}{e_i + 1}.$$

The first term is clearly nonnegative and the second one can be checked to be nonnegative for $e_i \in \{1,2,3,4\}$. Note that integrality of e_i plays a key role here, as the second term is negative for $e_i \in (2,3)$.

◀

We can show that the above bound is essentially tight

▶ **Theorem 3.9.** *For any $n \in \mathbb{N}, u^* \in \mathbb{R}_{\geq 0}$, there exists $C \in CONF(n,u^*)$ such that $val(C) = u^* + \frac{1}{6}(n-u^*) - O(1)$.*

Proof. It is quite easy to construct such C by looking at the proof of Theorem 3.8. We get the first tight example when, in Case 2 of the analysis, we have $u_i = 0$ and $e_i \in \{2,3\}$. This corresponds to configurations consisting of elements of the form:

- $x_i = \frac{2}{3}, l_i = \frac{4}{3}, u_i = 0$, in which case we have $e_i = 2$ and so $u_i + \frac{1}{6}(e_i - u_i) = \frac{1}{3}$ and $\mathrm{val}_i = l_i + u_i - 1 = \frac{1}{3}$, or
- $x_i = \frac{1}{2}, l_i = \frac{3}{2}, u_i = 0$, in which case we have $e_i = 3$ and so $u_i + \frac{1}{6}(e_i - u_i) = \frac{1}{2}$ and $\mathrm{val}_i = l_i + u_i - 1 = \frac{1}{2}$.

Using these two items we can construct tight examples for $u^* = 0$ and arbitrary $n \geq 2$.

To handle the case of $u^* > 0$ we need another (almost) tight case in the proof of Theorem 3.8 which occurs when u_i is close to e_i and e_i is relatively large. In this case the value of the expression $(e_i - u_i)\left(\frac{1}{6} - \frac{1}{1+e_i}\right) + \frac{1}{1+e_i}$ is clearly close to 0. This corresponds to using items of the form $x_i = 1, l_i = 1$ and arbitrary u_i. For such elements we have $e_i = \lceil u_i + 1 \rceil$ and so

$$u_i + \frac{1}{6}(e_i - u_i) \leq u_i + \frac{1}{3},$$

and

$$\mathrm{val}_i = l_i + u_i - 1 = u_i,$$

so the difference between the two is at most $\frac{1}{3}$. By combining the three types of items described, we can clearly construct C as required for any n and u^*. ◀

We are now ready to prove the Lemma 3.1.

Proof (of Lemma 3.1). It follows from Theorem 3.8 and Fact 3.6 that

$$|f'| \leq u^* + \frac{1}{6}(n-u^*) = \frac{5}{6}u^* + \frac{1}{6}n.$$

Using Fact 3.3 we get:

$$|f'| \leq \frac{5}{6} \cdot 2(\mathrm{OPT}_{LP} - n) + \frac{1}{6}n = \frac{5}{3}\mathrm{OPT}_{LP} - \frac{3}{2}n.$$

◀

4 New upper bound for $|f''|$

In this section we give a new bound for $|f''|$. We do not bound it directly, as in Lemma 2.7. Instead, we show the following.

▶ **Lemma 4.1.**

$$|f''| \leq \frac{5}{6}\left(2OPT_{LP}(G) - 2n - u^*\right).$$

What this says is basically that f'' can be fully paid for by ($\frac{5}{6}$ of) the slack we get in Fact 3.3. To better understand this bound, and in particular the constant $\frac{5}{6}$, before we proceed to prove it, let us first show how it can be used.

▶ **Corollary 4.2.** $|f| \leq \frac{5}{3}OPT_{LP} - \frac{3}{2}n$.

Proof. We have $|f| \leq |f'| + |f''| \leq \frac{5}{6}u^* + \frac{1}{6}n + \frac{5}{6}\left(2\mathrm{OPT}_{LP} - 2n - u^*\right) = \frac{5}{3}\mathrm{OPT}_{LP} - \frac{3}{2}n$. ◀

There are several interesting things to note here. First of all, we got the exact same bound as in Lemma 3.1, which means that $|f''|$ can be fully paid for by the slack in Fact 3.3, as suggested earlier. In particular, this means that improving the constant $\frac{5}{6}$ in Lemma 4.1 is pointless, since we would still be getting the same bound on $|f|$ when $|f''| = 0$. Therefore, we do not try to optimize this constant, but instead make the proof of the Lemma as straightforward as possible.

Let us now proceed to prove Lemma 4.1. For any non-root in-vertex w let $z_w = x^*_{t_w} + x^*(B(w))$. Basically, if v is the parent of w in $\vec{T}$, then z_w is the total value of x^* over all edges connecting v with vertices in the subtree T_w of T determined by w. Also, let γ_w be the total of x^* over all edges connecting vertices in T_w with vertices above v.

We can formulate the following local version of Lemma 4.1.

▶ **Lemma 4.3.** *For every non-root vertex v of G we have*

$$\sum_{w\in\mathcal{I}_v} \max(0, 1-\gamma_w) \leq \frac{5}{6}\left(x^*(v) - 2 - \sum_{w\in\mathcal{I}_v} u_w\right).$$

Notice that Lemma 4.1 easily follows from Lemma 4.3 by summing over all non-root vertices.

Proof (of Lemma 4.3). Let v be a non-root vertex of G. We define 3 types of vertices in $\mathcal{I}_v$:

- $w \in \mathcal{I}_v$ is *heavy* if $\gamma_w < 1$ and $z_w > 2$
- $w \in \mathcal{I}_v$ is *light* if $\gamma_w < 1$ and $z_w \leq 2$,
- $w \in \mathcal{I}_v$ is *trivial* otherwise (i.e. $\gamma_w \geq 1$).

We denote by H_v and L_v the sets of heavy and light vertices in $\mathcal{I}_v$, respectively. Intuitively, heavy vertices are the ones that contribute to both u^* and $|f''|$, light vertices contribute only to $|f''|$, and the remaining (i.e. trivial) vertices do not contribute to $|f''|$.

We are going to use the following observations:

1. $z_w \geq 2 - \gamma_w$ for all $w \in H_v \cup L_v$,
2. $x^*(v) \geq \sum_{w\in H_v\cup L_v} z_w + \max(0, 2 - \sum_{H_v\cup L_v} \gamma_w)$.

The first observation follows from the Held-Karp inequality for the cut induced by the subtree T_w of T determined by w. The second follows from Held-Karp inequality as well, this time for the cut induced by the set $\bigcup_{w\in H_v\cup L_v} T_w \cup \{v\}$. The only edges crossing this cut are the back-edges with total x^* value $\sum_{H_v\cup L_v} \gamma_w$, and edges incident to v, but not to a vertex from a subtree induced by one of $w \in H_v \cup L_v$. The second term in the second observation is a

lower-bound on the total x^* value of this second kind of edges, resulting from the Held-Karp inequality.

Note that the trivial vertices might have $z_w > 2$ and so contribute to u^*. However in that case the proof is quite simple and it will be advantageous for us to get it out of our way. Let w_0 be a trivial vertex with $z_{w_0} > 2$. What we do is basically use this vertex to cancel out the lone 2 in the second factor of the RHS of the bound in Lemma 4.3:

$$x^*(v) - 2 - \sum_{w\in\mathcal{I}_v} u_w \geq \sum_{w\in\mathcal{I}_v\setminus w_0} z_w + z_{w_0} - 2 - \sum_{w\in\mathcal{I}_v\setminus w_0} u_w - u_{w_0} = \sum_{w\in\mathcal{I}_v\setminus w_0} (z_w - u_w)$$

Since $w_0 \notin H_v \cup L_v$ we thus have

$$\frac{5}{6}\left(x^*(v) - 2 - \sum_{w\in\mathcal{I}_v} u_w\right) \geq \sum_{w\in H_v\cup L_v} \frac{5}{6}(z_w - u_w) \geq \sum_{w\in H_v\cup L_v} (1-\gamma_w).$$

The last inequality holds because we have $z_w - u_w = 2$ for heavy w and $z_w - u_w = z_w \geq 2 - \gamma_w$ for light w. We can thus assume that all trivial vertices have $z_w \leq 2$ (and so $u_w = 0$).

Note that using our observations, we can reformulate our claim as follows:

$$\sum_{w\in H_v\cup L_v} (1-\gamma_w) \leq \frac{5}{6}\left(\sum_{w\in H_v\cup L_v} z_w + \max\left(0, 2 - \sum_{w\in H_v\cup L_v} \gamma_w\right) - 2 - \sum_{w\in\mathcal{I}_v} u_w\right).$$

and since we now assume that trivial vertices have $z_w \leq 2$, it is enough to prove:

$$\sum_{w\in H_v\cup L_v} (1-\gamma_w) \leq \frac{5}{6}\left(\sum_{w\in L_v} z_w + \max\left(0, 2 - \sum_{w\in H_v\cup L_v} \gamma_w\right) + 2(|H_v| - 1)\right)$$

(since $z_w = 2 + u_w$ for $w \in H_v$).

Clearly, if all $w \in \mathcal{I}_v$ are trivial, both sides of the bound are 0 and so it trivially holds. Otherwise, we consider the following two cases:

Case 1: $\sum_{w\in H_v\cup L_v} \gamma_w > 2$. Notice that this implies $|H_v| + |L_v| \geq 3$. In this case the RHS of the bound becomes

$$\frac{5}{6}\left(\sum_{w\in L_v} z_w + 2(|H_v| - 1)\right) \geq \frac{5}{6}\left(\sum_{w\in L_v} (2-\gamma_w) + 2(|H_v| - 1)\right).$$

The ratio of the above expression and the LHS is lower-bounded by the ratio of these same expressions with all $\gamma_w = 0$, i.e. $\frac{5}{6} \cdot \frac{2(|L_v|+|H_v|-1)}{|L_v|+|H_v|}$, which is definitely at least 1, since $|L_v| + |H_v| \geq 3$.

Case 2: $\sum_{w\in H_v\cup L_v} \gamma_w \leq 2$. In this case the RHS of the bound becomes

$$\frac{5}{6}\left(\sum_{w\in L_v} z_w + 2 - \sum_{w\in H_v\cup L_v} \gamma_w + 2(|H_v| - 1)\right) \geq \frac{5}{6}\left(\sum_{w\in L_v} (2-2\gamma_w) + \sum_{w\in H_v} (2-\gamma_w)\right).$$

The claim now follows by observing that $(2 - 2\gamma_w) = 2(1-\gamma_w)$ and $2 - \gamma_w \geq 2(1-\gamma_w)$.

◀

5 Applications to graphic TSP and TSPP

As a consequence of Corollary 4.2, we get improved approximation factors for graphic TSP and graphic TSPP.

▶ **Theorem 5.1.** *There is a $\frac{13}{9}$-approximation algorithm for graphic TSP.*

Proof. By Corollary 4.2 we get

$$|f| \leq \frac{5}{3}\mathrm{OPT}_{LP} - \frac{3}{2}n.$$

The TSP tour guaranteed by Theorem 2.4 has size at most

$$\frac{4}{3}n + \frac{2}{3}|f| \leq \frac{4}{3}n + \frac{2}{3}\left(\frac{5}{3}\mathrm{OPT}_{LP} - \frac{3}{2}n\right) = \frac{10}{9}\mathrm{OPT}_{LP} + \frac{1}{3}n.$$

Notice that the approximation ratio of the resulting algorithm is getting better with OPT_{LP} increasing (with fixed n). Therefore the worst case bound is the one we get for $\mathrm{OPT}_{LP} = n$, i.e. $\frac{10}{9} + \frac{1}{3} = \frac{13}{9}$. ◀

▶ Remark. This analysis is significantly simpler than the one in [8]. Balancing with Christofides's algorithm is no longer necessary since bounds on approximation ratios for both algorithms are decreasing in OPT_{LP}.

▶ **Theorem 5.2.** *There is a $\frac{19}{12} + \varepsilon$-approximation algorithm for graphic TSPP, for any $\varepsilon > 0$.*

Proof. This proof is very similar to the proof of Theorem 1.2 in [8]. However, the reasoning is slighly simpler, in our opinion. Suppose we want to approximate the graphic TSPP in $G = (V, E)$ with end-vertices s and t. Let $G' = (V, E \cup \{e'\})$, where $e' = \{s, t\}$, and let OPT_{LP} denote $\mathrm{OPT}_{LP}(G')$. Also, let d be the distance between s and t in G. By Corollary 4.2 we get

$$|f| \leq \frac{5}{3}\mathrm{OPT}_{LP} - \frac{3}{2}n.$$

The TSP path guaranteed by Theorem 2.5 has size at most

$$\frac{4}{3}n + \frac{2}{3}|f| - \frac{2}{3} + \frac{d}{3} \leq \frac{4}{3}n + \frac{2}{3}\left(\frac{5}{3}\mathrm{OPT}_{LP} - \frac{3}{2}n\right) - \frac{2}{3} + \frac{d}{3} = \frac{10}{9}\mathrm{OPT}_{LP} + \frac{n+d-2}{3}.$$

It is clear that the quality of this algorithm deteriorates as d increases. We are going to balance it with another algorithm that displays the opposite behaviour. The following approach is folklore: Find a spanning tree T in G and double all edges of T except those that lie on the unique shortest path connecting s and t. The resulting graph has a spanning Eulerian path connecting s and t with at most $2(n-1) - d$ edges.

Since $\mathrm{OPT}_{LP} - 1 \leq \mathrm{OPT}_{LP}(G, s, t)$ is a lower bound for the optimal solution, the two approximation algorithms have approximation ratios bounded by

$$\frac{\frac{10}{9}\mathrm{OPT}_{LP} + \frac{n+d-2}{3}}{\mathrm{OPT}_{LP} - 1}$$

and

$$\frac{2n-2-d}{\mathrm{OPT}_{LP} - 1}.$$

For a fixed value of OPT_{LP} the first of these expressions is increasing and the second is decreasing in d. Therefore the worst case bound we get for an algorithm that picks the best of the two solutions occurs when

$$\frac{10}{9}\mathrm{OPT}_{LP} + \frac{n+d-2}{3} = 2n - 2 - d,$$

which leads to

$$d = \frac{5}{4}n - \frac{5}{6}\mathrm{OPT}_{LP} - 1.$$

For this value of d the approximation ratio is at most

$$\frac{2n-2-\left(\frac{5}{4}n - \frac{5}{6}\mathrm{OPT}_{LP} - 1\right)}{\mathrm{OPT}_{LP}-1} = \frac{\frac{3n}{4} - 1 + \frac{5}{6}\mathrm{OPT}_{LP}}{\mathrm{OPT}_{LP}-1} = \frac{\frac{3n}{4} - \frac{1}{6}}{\mathrm{OPT}_{LP}-1} + \frac{5}{6}.$$

Since $\mathrm{OPT}_{LP} \geq n$ this is at most

$$\frac{\frac{3n}{4} - \frac{1}{3}}{n-1} + \frac{5}{6} = \frac{3}{4} + \frac{5}{6} + O\left(\frac{1}{n}\right) = \frac{19}{12} + O\left(\frac{1}{n}\right),$$

which proves the claim. ◀

▶ Remark. One might ask why the improvement for the graphic TSP is much bigger than the one for graphic TSPP. The reason for that is that while for large values of OPT/n our bound on $|f|$ is significantly better than the one in [8], it is only slightly better when $\mathrm{OPT} = n$. As it turns out, this is exactly the worst case for TSPP, both in our analysis and in the one in [8]. For TSP however, the worst case value of OPT for the analysis in [8] is larger than n.

6 Acknowledgements

The author would like to thank Marcin Pilipczuk for pointing out some problems in an early version of this work, and an anonymous referee for his comments, which greatly improved the presentation.

References

1 Nicos Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, CMU, 1976.
2 G. Cornuejols, D. Naddef, and J. Fonlupt. The traveling salesman problem on a graph and some related integer polyhedra. *Math Programming*, 33:1–27, 1985.
3 Shayan Oveis Gharan, Amin Saberi, and Mohit Singh. A randomized rounding approach to the travelling salesman problem. In *FOCS'11 (to appear)*, 2011.
4 Michel X. Goemans and Dimitris J. Bertsimas. On the parsimonious property of connectivity problems. In *SODA'90*, pages 388–396, 1990.
5 Michelangelo Grigni, Elias Koutsoupias, and Christos H. Papadimitriou. An approximation scheme for planar graph tsp. In *FOCS'95*, pages 640–645, 1995.
6 Michael Held and Richard M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.
7 J. A. Hoogeveen. Analysis of christofides' heuristic: some paths are more difficult than cycles. *Operations Research Letters*, 10(5):291—-295, 1991.
8 Tobias Mömke and Ola Svensson. Approximating graphic tsp by matchings. In *FOCS'11 (to appear)*, 2011.
9 David B. Shmoys and David P. Williamson. Analyzing the held-karp tsp bound: a monotonicity property with application. *Information Processing Letters*, 35(6):281 – 285, 1990.

A $(k+3)/2$-approximation algorithm for monotone submodular k-set packing and general k-exchange systems

Justin Ward

Department of Computer Science, University of Toronto
Toronto, Canada
jward@cs.toronto.edu

Abstract

We consider the monotone submodular k-set packing problem in the context of the more general problem of maximizing a monotone submodular function in a k-exchange system. These systems, introduced by Feldman et al. [9], generalize the matroid k-parity problem in a wide class of matroids and capture many other combinatorial optimization problems. We give a deterministic, non-oblivious local search algorithm that attains an approximation ratio of $(k+3)/2+\epsilon$ for the problem of maximizing a monotone submodular function in a k-exchange system, improving on the best known result of $k+\epsilon$, and answering an open question posed by Feldman et al.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases k-set packing, k-exchange systems, submodular maximization, local search, approximation algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.42

1 Introduction

In the general k-set packing problem, we are given a collection $\mathcal{G}$ of sets, each with at most k elements, and an objective function $f : 2^{\mathcal{G}} \to \mathbb{R}_+$ assigning each subset of $\mathcal{G}$ a value and seek a collection $S \subseteq \mathcal{G}$ of pairwise-disjoint sets maximizing f. In the special case that $f(A) = |A|$, we obtain the *unweighted k-set packing problem*. Similarly, if f is linear function, so that $f(A) = \sum_{e \in A} w(e)$ for some weight function $w : \mathcal{G} \to \mathbb{R}_+$ we obtain the *weighted k-set packing problem*. In this paper we consider the case in which f may be any monotone submodular function.

For unweighted k-set packing, Hurkens and Schrijver [15] and Halldórsson [13] independently obtained a $k/2+\epsilon$ approximation via a simple local search algorithm. Using similar techniques, Arkin and Hassin [1] obtained a $k-1+\epsilon$ approximation for weighted k-set packing, and showed that this result is tight for their simple local search algorithm. Chandra and Halldórsson [5] showed that a more sophisticated local search algorithm, which starts with a greedy solution and always chooses the best possible local improvement at each stage, attains an approximation ratio of $2(k+1)/3+\epsilon$. This was improved further by Berman [2], who gave a *non-oblivious* local search algorithm yielding a $(k+1)/2+\epsilon$ approximation for weighted k-set packing. Non-oblivious local search [16] is a variant of local search in which an auxiliary objective function, rather than the problem's given objective, is used to evaluate solutions. In the case of Berman, the local search procedure repeatedly seeks to improve the sum of the *squares* of the weights in the current solution, rather than the sum of the weights.

Many of the above local search algorithms for k-set packing yield the same approximations for the more general problem of finding maximum independent sets in $(k+1)$-claw free graphs.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 42–53

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


Local search techniques have also proved useful for other generalizations of k-set packing, including variants of the matroid k-parity problem [18, 20]. Motivated by the similarities between these problems, Feldman et al. [9] introduced the class of *k-exchange* systems, which captures many problems amenable to approximation by local search algorithms. These systems are formulated in the general language of independence systems, which we now briefly review.

An independence system is specified by a ground set $\mathcal{G}$, and a hereditary (i.e. non-empty and downward-closed) family $\mathcal{I}$ of subsets of $\mathcal{G}$. The sets in $\mathcal{I}$ are called *independent sets*, and the inclusion-wise maximal sets in $\mathcal{I}$ are called *bases*. Given an independence system $(\mathcal{G}, \mathcal{I})$ and a function $f : 2^{\mathcal{G}} \to \mathbb{R}_+$, we shall consider the problem of finding an independent set $S \in \mathcal{I}$ that maximizes f.

The class of k-exchange systems satisfy the following additional property:

▶ **Definition 1** (k-exchange system [9]). An independence system $(\mathcal{G}, \mathcal{I})$ is a *k-exchange system* if, for all A and B in $\mathcal{I}$, there exists a multiset $Y = \{Y_e \subseteq B \setminus A \mid e \in A \setminus B\}$, containing a subset Y_e of $B \setminus A$ for each element $e \in A \setminus B$, that satisfies:

(K1) $|Y_e| \leq k$ for each $x \in A$.
(K2) Every $x \in B \setminus A$ appears in at most k sets of Y.
(K3) For all $C \subseteq A \setminus B$, $(B \setminus (\bigcup_{e \in C} Y_e)) \cup C \in \mathcal{I}$.

We call the set Y_e in Definition 1 the *neighborhood* of e in B. For convenience, we extend the collection Y in Definition 1 by including the set $Y_x = \{x\}$ for each element $x \in A \cap B$. It is easy to verify that the resulting collection still satisfies conditions (K1)–(K3).

The 1-exchange systems are precisely the class of strongly base orderable matroids described by Brualdi [3]. This class is quite large and includes all gammoids, and hence all transversal and partition matroids. For $k > 1$, the class of k-exchange systems may be viewed as a common generalization of the matroid k-parity problem in strongly base orderable matroids and the independent set problem in $(k+1)$-claw free graphs. Feldman et al. showed that k-exchange systems encompass a wide variety of combinatorial optimization problems, including k-set packing, intersection of k strongly base orderable matroids, b-matching (here $k = 2$), and asymmetric traveling salesperson (here $k = 3$).

Our results hold for any k-exchange system, and so we present them in the general language of Definition 1. However, the reader may find it helpful to think in terms of a concrete problem, such as the k-set packing problem. In that case, the ground set $\mathcal{G}$ is the given collection of sets, and a sub-collection of sets $S \subseteq \mathcal{G}$ is independent if and only if all the sets in S are disjoint. Given A and B as in Definition 1, Y_e is the set of all sets in B that contain any element contained by the set $e \in A$ (i.e. the set of all sets in B that are not disjoint from e). Then, property (K3) is immediate, and (K1) and (K2) follow directly from the fact that each set in $\mathcal{G}$ contains at most k elements.

1.1 Related Work

Recently, the problem of maximizing submodular functions subject to various constraints has attracted much attention. We focus here primarily on results pertaining to matroid constraints and related independence systems.

In the case of an arbitrary single matroid constraint, Calinescu et al. have attained a $e/(e-1)$ approximation for monotone submodular maximization via the *continuous greedy algorithm*. This result is tight, provided that $P \neq NP$ [6]. In the case of $k \geq 2$ simultaneous matroid constraints, an early result of Fisher, Nemhauser, and Wolsey [10] shows that

the standard greedy algorithm attains a $k+1$ approximation for monotone submodular maximization. Fischer et al. state further that the result can be generalized to k-systems (a full proof appears in Calinescu et al. [4]). More recently, Lee, Sviridenko, and Vondràk [19] have improved this result to give a $k+\epsilon$ approximation for monotone submodular maximization over $k \geq 2$ arbitrary matroid constraints via a simple, oblivious local search algorithm. Feldman et al. [9] used a similar analysis to show that oblivious local search attains a $k+\epsilon$ approximation for the class of k-exchange systems (here, again, $k \geq 2$). For the more general class of k-systems, Gupta et al. [12] give a $(1+\beta)(k+2+1/k)$ approximation, where β is the best known approximation ratio for unconstrained non-monotone submodular maximization.

In the case of unconstrained non-monotone submodular maximization, Feige, Mirrokni, and Vondrák [7] gave a randomized 2.5 approximation, which was iteratively improved by Gharan and Vondrák [11] and then Feldman, Naor, and Shwartz [8] to ≈ 2.38. For non-monotone maximization subject to k matroid constraints, Lee, Sviridenko, and Vondrák [17] gave a $k+2+1/k+\epsilon$ approximation, and later improved [19] this to a $k+1+1/(k-1)+\epsilon$ approximation. Again, the latter result is obtained by a standard local search algorithm. Feldman et al. [9] apply similar techniques to yield a $k+1+1/(k-1)+\epsilon$ approximation for non-monotone submodular maximization the general class of k-exchange systems.

1.2 Our Contribution

In the restricted case of a linear objective function, Feldman et al. [9] gave a non-oblivious local search algorithm inspired by Berman's algorithm [2] for $(k+1)$-claw free graphs. They showed that the resulting algorithm is a $(k+1)/2+\epsilon$ approximation for linear maximization in any k-exchange system. Here we consider a question posed in [9]: namely, whether a similar technique can be applied to the case of monotone submodular maximization in k-exchange systems. In this paper, we answer this question affirmatively, giving a non-oblivious local search algorithm for monotone submodular maximization in a k-exchange system. As in [9], the k-exchange property is used only in the analysis of our algorithm. Our algorithm attains an approximation ratio of $\frac{k+3}{2}+\epsilon$. For $k>3$, this improves upon the $k+\epsilon$ approximation obtained by the oblivious local search algorithm presented in [9]. Additionally, we note that our algorithm runs in time polynomial in ϵ^{-1}, while the $k+\epsilon$ approximation algorithm of [9] requires time exponential in ϵ^{-1}.

As a consequence of our general result, we obtain an improved approximation guarantee of $\frac{k+3}{2}$ for a variety of monotone submodular maximization problems (some of which are generalizations of one another) including: k-set packing, independent sets in $(k+1)$-claw free graphs, k-dimensional matching, intersection of k strongly base orderable matroids, and matroid k-parity in a strongly base orderable matroid. In all cases, the best previous result was $k+\epsilon$.

2 A First Attempt at the Submodular Case

Before presenting our algorithm, we describe some of the difficulties that arise when attempting to adapt the non-oblivious local search algorithm of [2] and [9] to the submodular case. Our hope is that this will provide some intuition for our algorithm, which we present in the next section.

We recall that a function $f: 2^{\mathcal{G}} \to \mathbb{R}_+$ is submodular if $f(A)+f(B) \geq f(A \cup B)+f(A \cap B)$ for all $A, B \subseteq \mathcal{G}$. Equivalently, f is submodular if for all $S \subseteq T$ and all $x \notin T$, $f(S+x)-f(S) \geq f(T+x)-f(T)$. In other words, submodular functions are characterized by decreasing

marginal gains. We say that a submodular function f is monotone if it additionally satisfies $f(S) \leq f(T)$ for all $S \subseteq T$.

The non-oblivious algorithm of [9] for the linear case is shown in Algorithm 1. It repeatedly searches for a k-replacement (A, B) that improves the non-oblivious potential function w^2. Formally, we call the pair of sets (A, B), where $B \subseteq S$ and $A \subseteq \mathcal{G} \setminus (S \setminus B)$ a *k-replacement* if $|A| \leq k$, $|B| \leq k^2 - k + 1$ and $(S \setminus B) \cup A \in \mathcal{I}$. If $w(e) = f(\{e\})$ is the weight assigned to an element e, then the non-oblivious potential function used by Algorithm 1 is given by $w^2(S) = \sum_{e \in S} w(e)^2$. That is, our non oblivious potential function $w^2(S)$ is simply the sum of the *squared* weights of the elements of S.[1] We use the fact that $w^2(S) > w^2((S \setminus B) \cup A)$ if and only if $w^2(A) > w^2(B)$, to slightly simplify the search for an improvement.

Algorithm 1: Non-Oblivious Local Search for Linear Objective Functions

Input:
- Ground set $\mathcal{G}$
- Membership oracle for $\mathcal{I} \subseteq 2^{\mathcal{G}}$
- Value oracle for monotone submodular function $f : 2^{\mathcal{G}} \to \mathbb{R}_+$
- Approximation parameter $\epsilon \in (0, 1)$

Let $S_{init} = \{\arg\max_{e \in \mathcal{G}} w(e)\}$;
Let $\alpha = w(S_{init})\epsilon/n$;
Round all weights $w(e)$ down to integer multiples of α;
$S \leftarrow S_{init}$;
$S_{old} \leftarrow S$;
repeat
  foreach k-replacement (A, B) **do**
    if $w^2(A) > w^2(B)$ **then**
      $S_{old} \leftarrow S$;
      $S \leftarrow (S \setminus B) \cup A$;
      break;
until $S_{old} = S$;
return S;

In the monotone submodular case, we can no longer necessarily represent f as a sum of weights. However, borrowing some intuition from the greedy algorithm, we might decide to replace each weight $w(e)$ in the potential function w with the marginal gain/loss associated with e. That is, at the start of each iteration of the local search algorithm, we assign each element $e \in \mathcal{G}$ weight $w(e) = f(S + e) - f(S - e)$, where S is the algorithm's current solution, then proceed as before. Note that $w(e)$ is simply the marginal gain attained by adding e to S (in the case that $e \notin S$) or the marginal loss suffered by removing e from S (in the case that $e \in S$). We define the non-oblivious potential function w^2 in terms of the resulting weight function w as before.

Unfortunately, the resulting algorithm may fail to terminate, as the following small example shows. We consider a simple, unweighted coverage function on the universe $U =$

[1] To ensure polynomial-time convergence, Algorithm 1 first round the weights down to integer multiples of a suitable small value α, related to the approximation parameter ϵ. The algorithm then converges in time polynomial in ϵ^{-1} and n, at a loss of only $(1 - \epsilon)^{-1}$ in the approximation factor.

$\{a, b, c, x, y, z\}$. Let:

$$S_1 = \{a, b\} \qquad S_3 = \{x, y\}$$
$$S_2 = \{a, c\} \qquad S_4 = \{x, z\}$$

Our ground set $\mathcal{G}$ is then $\{1, 2, 3, 4\}$ and our objective function $f(A) = \left|\bigcup_{i \in A} S_i\right|$ for all $A \subseteq \mathcal{G}$. We consider the 2-exchange system with only 2 bases: $P = \{1, 2\}$ and $Q = \{3, 4\}$. For current solution $S = P$ we have $w(1) = w(2) = 1$ and $w(3) = w(4) = 2$. Since $w^2(\{1, 2\}) = 2 < 8 = w^2(\{3, 4\})$, the 2-replacement $(\{3, 4\}, \{1, 2\})$ is applied, and the current solution becomes Q. In the next iteration, we have $S = Q$, and $w(1) = w(2) = 2$ and $w(3) = w(4) = 1$, so the 2-replacement $(\{1, 2\}, \{3, 4\})$ is applied by the algorithm. This returns us to the solution to P, where the process repeats indefinitely.

3 The New Algorithm

Intuitively, the problem with this initial approach is that the weight function used at each step of the algorithm depends on the current solution S (since all marginals are taken with respect to S). Hence, it may be the case that a k-replacement (A, B) results in an improvement with respect to the current solution's potential function, but in fact results in a *decreased* potential value in the next iteration after the weights have been updated. Surprisingly, we can solve the problem by introducing *even more* variation in the potential function. Specifically, we allow the algorithm to use a different weight function not only for each current solution S, but also for *each k-replacement (A, B) that is considered*. We give the full algorithm at the end of this section and a detailed analysis in the next.

First, we describe the general approach we use to generate the weights used in our potential function. Rather than calculating all marginal gains with respect to S, we consider elements in some order and assign each element a weight corresponding to its marginal gain with respect to those elements that precede it. By carefully updating both the current solution and its order each time we apply a local improvement, we ensure that the algorithm converges to a local optimum.

The algorithm stores the current solution S as an ordered sequence $s_1, s_2, \ldots, s_{|S|}$. At each iteration of the local search, before searching for an improving k-replacement, it assigns a weight $w(s_i)$ to each $s_i \in S$, as follows. Let $S_i = \{s_j \in S : j \leq i\}$ be the set containing the first i elements of S. Then, the weight function w assigning weights to the elements of S is given by

$$w(s_i) = f(S_{i-1} + s_i) - f(S_{i-1}) = f(S_i) - f(S_{i-1})$$

for all $s_i \in S$. Note that our weight function satisfies

$$\sum_{s_i \in S} w(s_i) = \sum_{i=1}^{|S|} [f(S_i) - f(S_{i-1})] = f(S) - f(\emptyset) \leq f(S) \ . \tag{1}$$

In order to evaluate a k-replacement (A, B), we also need to assign weights to the elements in $A \subseteq \mathcal{G} \setminus (S \setminus B)$. We use a different weight function for each k-replacement (A, B), obtained as follows. We order A according to an arbitrary ordering $\prec$ on $\mathcal{G}$ and a_i be the ith element of A and $A_i = \{a_j \in A : j \leq i\}$. Then, the weight function $w_{(A,B)}$ assigning weights to the elements of A is given by

$$w_{(A,B)}(a_i) = f((S \setminus B) \cup A_{i-1} + a_i) - f((S \setminus B) \cup A_{i-1}) = f((S \setminus B) \cup A_i) - f((S \setminus B) \cup A_{i-1})$$

for all $a_i \in A$. Note that for every k-replacement (A, B),

$$\sum_{x \in A} w_{(A,B)}(a_i) \geq \sum_{i=1}^{|A|} [f((S \setminus B) \cup A_i) - f((S \setminus B) \cup A_{i-1})]$$
$$= f((S \setminus B) \cup A) - f(S \setminus B) \geq f(S \cup A) - f(S) \ , \quad (2)$$

where the last inequality follows from the submodularity of f. Note that since the function f is *monotone* submodular, all of the weights w and $w_{(A,B)}$ that we consider will be non-negative. This fact plays a crucial role in our analysis. Furthermore, the weights w assigned to elements in S remain fixed for all k-replacements considered in a single phase of the algorithm. Finally, we note that although both w and $w_{(A,B)}$ depend on the current solution S, we omit this dependence from our notation, instead stating explicitly when there is a chance of confusion which solution's weight function we are considering.

Our final algorithm appears in Algorithm 2. We start from an initial solution $S_{init} = \arg\max_{e \in \mathcal{G}} f(\{e\})$, consisting of the single element of largest value. When applying a k-replacement (A, B), the algorithm updates the ordered solution S in a fashion that ensures all of the elements of $S \setminus B$ precede those of A, while all elements of $S \setminus B$ and, respectively, A occur in the same relative order. As we shall see in the next section, this guarantees that the algorithm will converge to a local optimum. As in the linear case, we use the sum of squared weights $w^2(B) = \sum_{b \in B} w(b)^2$ and $w^2_{(A,B)}(A) = \sum_{a \in A} w_{(A,B)}(a)^2$ to guide the search.

To ensure polynomial-time convergence, we round all of our weights down to the nearest integer multiple of α, which depends on the parameter ϵ. This ensures that every improvement improves the current solution by an additive factor of at least α^2. Because of this rounding factor, we must actually work with the following analogs of (1) and (2):

$$\sum_{x \in S} w(x) \leq \sum_{i=1}^{|S|} [f(S_i) - f(S_{i-1})] = f(S) - f(\emptyset) \leq f(S) \quad (3)$$

$$\sum_{x \in A} w_{(A,B)}(x) \geq \sum_{i=1}^{|A|} [f((S \setminus B) \cup A_i) - f((S \setminus B) \cup A_{i-1}) - \alpha]$$
$$= f((S \setminus B) \cup A) - f(S \setminus B) - |A|\alpha \geq f(S \cup A) - f(S) - |A|\alpha \quad (4)$$

4 Analysis of Algorithm 2

We now analyze the approximation and runtime performance of Algorithm 2. We consider the worst-case ratio, or *locality gap*, $f(O)/f(S)$ where S is any locally optimal solution (with respect to Algorithm 2's potential function) and O is a globally optimal solution. We shall need the following technical lemma, which is a direct consequence of Lemma 1.1 in [19]. We give a proof here for the sake of completeness.

▶ **Lemma 2.** *Let f be a submodular function on $\mathcal{G}$, Let $T, S \subseteq \mathcal{G}$, and $\{T_i\}_{i=1}^t$ be a partition of T. Then,*

$$\sum_{i=1}^{t} [f(S \cup T_i) - f(S)] \geq f(S \cup T) - f(S)$$

Algorithm 2: Non-Oblivious Local Search

Input:
- Ground set $\mathcal{G}$
- Membership oracle for $\mathcal{I} \subseteq 2^{\mathcal{G}}$
- Value oracle for monotone submodular function $f : 2^{\mathcal{G}} \to \mathbb{R}_+$
- Approximation parameter $\epsilon \in (0, 1)$

Fix an arbitrary ordering $\prec$ on the elements of $\mathcal{G}$
Let $S_{init} = \arg\max_{e \in \mathcal{G}} f(\{e\})$
Let $\delta = \left(1 + \frac{k+3}{2\epsilon}\right)^{-1}$ and $\alpha = f(S_{init})\delta/n$
$S \leftarrow S_{init}$ and $S_{old} \leftarrow S$
repeat
  $X \leftarrow \emptyset$
  for $i = 1$ **to** $|S|$ **do**
    $w(s_i) \leftarrow \lfloor (f(X + s_i) - f(X))/\alpha \rfloor \alpha$
    $X \leftarrow X + s_i$
  foreach k-replacement (A, B) **do**
    Let a_i be the ith element in A according to $\prec$
    $X \leftarrow S \setminus B$
    for $i = 1$ **to** $|A|$ **do**
      $w_{(A,B)}(a_i) \leftarrow \lfloor (f(X + a_i) - f(X))/\alpha \rfloor \alpha$
      $X \leftarrow X + a_i$
    if $w^2_{(A,B)}(A) > w^2(B)$ **then**
      $S_{old} \leftarrow S$
      Delete all elements in B from S
      Append the elements of A to the end of S, in the order given by $\prec$.
      break
until $S_{old} = S$
return S

Proof. Define $A_0 = S$ and then $B_i = T_i \setminus S$ and $A_i = B_i \cup A_{i-1}$, for all $1 \le i \le t$. Then, since $S \subseteq A_{i-1}$ for all $1 \le i \le t$, submodularity of f implies

$$f(T_i \cup S) - f(S) = f(B_i \cup S) - f(S) \le f(B_i \cup A_{i-1}) - f(A_{i-1}) = f(A_i) - f(A_{i-1})$$

for all $1 \le i \le t$. Now, we have

$$\sum_{i=1}^{t} [f(S \cup T_i) - f(S)] \ge \sum_{i=1}^{t} [f(A_i) - f(A_{i-1})] = f(A_t) - f(A_0) = f(S \cup T) - f(S) \; . \quad \blacktriangleleft$$

4.1 Approximation Ratio of Algorithm 2

We now consider the approximation ratio of Algorithm 2. A general outline of the proof is as follows: we consider only a particular set of k-replacements (A, B) and derive (in Lemma 3) a relationship between the non-oblivious potential functions w^2 and $w^2_{(A,B)}$ and the weight functions w and $w_{(A,B)}$ for these k-replacements. We use this relationship to derive (in Lemma 4) a lower bound on the weight $w(x)$ of each element x in a locally optimal solution. Finally, in Theorem 5 we combine these lower bounds to obtain a bound on the locality gap of Algorithm 2.

For the rest of this subsection, we consider an arbitrary instance $(\mathcal{G}, \mathcal{I}, f)$ and suppose that S is a locally optimal solution returned by the algorithm on this instance, while O is a global optimum for this instance. Because S is locally optimal, we must have $w^2_{(A,B)}(A) \leq w^2(B)$ for every k-replacement (A, B), where w and each $w_{(A,B)}$ are the weight functions determined by S.

We now describe the set of k-replacements used in our analysis. We have $S, O \in \mathcal{I}$ for the k-exchange system $(\mathcal{G}, \mathcal{I})$. Thus, there must be a collection Y assigning each e of O a neighborhood $Y_e \subseteq S$, satisfying the conditions of Definition 1. For each $x \in S$, let P_x be the set of all elements $e \in O$ for which: (1) $x \in Y_e$ and (2) for all $z \in Y_e$, $w(z) \leq w(x)$. That is, P_x is the set of all elements of O which have x as their heaviest neighbor in S. Note that the construction of P_x depends on the weights w assigned to elements in S being fixed throughout each iteration and independent of the particular improvement under consideration.

We define $N_x = \bigcup_{e \in P_x} Y_e$, and consider (P_x, N_x). From property (K2) of Y we have $|P_x| \leq k$. Similarly, from property (K1) and the fact that all elements $e \in P_x$ have as a common neighbor $x \in Y_e$ we have $|N_x| \leq 1 + k(k-1) = k^2 - k + 1$. Finally, from property (K3) we have $(S \setminus N_x) \cup P_x \in \mathcal{I}$. Thus, (P_x, N_x) is a valid k-replacement for all of our sets P_x, $x \in S$. Furthermore, $\{P_x\}_{x \in S}$ is a partition of O, and by the definition of P_x, we have $w(x) \geq w(z)$ for all $z \in N_x$. Again, this depends on the weights of elements in S being the same for all k-replacements considered by the algorithm during a given phase.

The following extension of a theorem from [2], relates the squared weight potentials w^2 and $w^2_{(P_x,N_x)}$ to the weight functions w and $w_{(P_x,N_x)}$ for each of our k-replacements (P_x, N_x).

▶ **Lemma 3.** *For all $x \in S$, and $e \in P_x$,*

$$w_{(P_x,N_x)}(e)^2 - w^2(Y_e - x) \geq w(x) \cdot \left(2w_{(P_x,N_x)}(e) - w(Y_e)\right) \quad .$$

Proof. First, we note that

$$0 \leq (w(x) - w_{(P_x,N_x)}(e))^2 = w(x)^2 - 2w(x) \cdot w_{(P_x,N_x)}(e) + w_{(P_x,N_x)}(e)^2 \quad . \tag{5}$$

Additionally, since $e \in P_x$, every element z in Y_e has weight at most $w(x)$, and so

$$w^2(Y_e - x) = \sum_{z \in Y_e - x} w(z)^2 \;\leq\; w(x) \sum_{z \in Y_e - x} w(z) \;=\; w(x) \cdot w(Y_e - x) \quad . \tag{6}$$

Adding (5) and (6) then rearranging terms using $w(x) \cdot w(Y_e - x) + w(x)^2 = w(x) \cdot w(Y_e)$ gives the desired result. ◀

We now prove the following lemma, which uses the local optimality of S to obtain a lower bound on the weight $w(x)$ of each element $x \in S$.

▶ **Lemma 4.** *For each $x \in S$, $w(x) \geq \sum_{e \in P_x} \left[2w_{(P_x,N_x)}(e) - w(Y_e)\right]$.*

Proof. Because S is locally optimal with respect to k-replacements, including in particular (P_x, N_x), we must have

$$w^2_{(P_x,N_x)}(P_x) \leq w^2(N_x) \quad . \tag{7}$$

First, we consider the case $w(x) = 0$. Recall that all the weights produced by the algorithm are non-negative, and $w(x)$ is the largest weight in N_x. Thus, $w(e) = 0$ for all $e \in N_x$ and $w^2(N_x) = 0$. Moreover, (7) implies that $w^2_{(P_x,N_x)}(P_x) = 0$ as well, and so $w_{(P_x,N_x)}(e) = 0$ for all $e \in P_x$. The claim then follows.

Now, suppose that $w(x) \neq 0$, and so $w(x) > 0$. From (7), together with the fact that $x \in Y_e$ for all $e \in P_x$, we have:

$$w^2_{(P_x,N_x)}(P_x) \leq w^2(N_x) \leq w(x)^2 + \sum_{e \in P_x} w^2(Y_e - x) \ . \tag{8}$$

Rearranging (8) using $w^2_{(P_x,N_x)}(P_x) = \sum_{e \in P_x} w_{(P_x,N_x)}(e)^2$ we obtain:

$$\sum_{e \in P_x} \left[w_{(P_x,N_x)}(e)^2 - w^2(Y_e - x) \right] \leq w(x)^2 \ . \tag{9}$$

Applying Lemma 3 to each term on the left of (9) we have:

$$\sum_{e \in P_x} w(x) \cdot \left[2w_{(P_x,N_x)}(e) - w(Y_e) \right] \leq w(x)^2 \ . \tag{10}$$

Dividing by $w(x)$ (recall that $w(x) \neq 0$) then yields

$$\sum_{e \in P_x} \left[2w_{(P_x,N_x)}(e) - w(Y_e) \right] \leq w(x) \ . \qquad \blacktriangleleft$$

We now prove our main result, which gives an upper bound on the locality gap of Algorithm 2.

▶ **Theorem 5.** $\left(\frac{k+3}{2} + \epsilon\right) f(S) \geq f(O)$

Proof. Lemma 4 gives us one inequality for each $x \in S$. We now add all $|S|$ inequalities to obtain

$$\sum_{x \in S} \sum_{e \in P_x} \left[2w_{(P_x,N_x)}(e) - w(Y_e) \right] \leq \sum_{x \in S} w(x) \ . \tag{11}$$

We have $\sum_{x \in S} w(x) \leq f(S)$ by (3). Additionally, from (4), $f(S \cup P_x) - f(S) - |P_x|\alpha \leq \sum_{e \in P_x} w_{(P_x,N_x)}(e)$ for every P_x. Thus, (11) implies

$$2 \sum_{x \in S} \left[f(S \cup P_x) - f(S) - |P_x|\alpha \right] - \sum_{x \in S} \sum_{e \in P_x} w(Y_e) \leq f(S) \ . \tag{12}$$

Since P is a partition of O, (12) is equivalent to

$$2 \sum_{x \in S} \left[f(S \cup P_x) - f(S) \right] - 2|O|\alpha - \sum_{e \in O} w(Y_e) \leq f(S) \ . \tag{13}$$

We have $w(x) \geq 0$ for all $x \in S$, and there are at most k distinct e for which $x \in Y_e$, by property (K2) of Y. Thus

$$\sum_{e \in O} w(Y_e) \leq k \sum_{x \in S} w(x) \leq k f(S) \ ,$$

by (3). Combining this with (13), we obtain

$$2 \sum_{x \in S} \left[f(S \cup P_x) - f(S) \right] - 2|O|\alpha - k f(S) \leq f(S) \ . \tag{14}$$

Using again the fact that P is a partition of O, we can apply Lemma 2 to the remaining sum on the left of 14, yielding

$$2 \left[f(S \cup O) - f(S) \right] - 2|O|\alpha - k f(S) \leq f(S) \ ,$$

which simplifies to

$$f(S \cup O) - |O|\alpha \leq \frac{k+3}{2} f(S) \ . \tag{15}$$

From the definition of α and the optimality of O, we have

$$|O|\alpha \leq n\alpha = \delta f(S_{init}) \leq \delta f(O) \ .$$

Finally, since f is monotone, we have $f(O) \leq f(S \cup O)$. Thus, (15) implies

$$(1-\delta) f(O) \leq \frac{k+3}{2} f(S) \ ,$$

which is equivalent to $f(O) \leq \left(\frac{k+3}{2} + \epsilon\right) f(S)$ by the definition of δ. ◀

4.2 Runtime of Algorithm 2

Now, we consider the runtime of Algorithm 2. Each iteration requires time $O(n)$ to compute the weights for S, plus time to evaluate all potential k-replacements. There are $O(n^{k+k(k-1)+1}) = O(n^{k^2+1})$ such k-replacements (A, B), and each one can be evaluated in time $O(k^2)$, including the computation of the weights $w_{(A,B)}$. Thus, the total runtime of Algorithm 2 is $O(Ik^2 n^{k^2+1})$, where I is the number of improvements it makes. The main difficulty remaining in our analysis is showing that Algorithm 2 constantly improves some global quantity, and so I is bounded. Here, we show that although the weights w assigned to elements of the current solution S change at each iteration, the non-oblivious potential $w^2(S)$, is monotonically increasing.

▶ **Lemma 6.** *Suppose that Algorithm 2 applies a k-replacement (A, B) to solution S to obtain a new solution T. Let w_S be the weight function w determined by solution S and w_T be the weight function w determined by solution T. Then, $w_T^2(T) \geq w_S^2(S) + \alpha^2$.*

Proof. We first show that $w_S(s_i) \leq w_T(s_i)$ for each element $s_i \in S \setminus B$ and $w_{(A,B)}(a_i) \leq w_T(a_i)$ for any element $a_i \in A$. In the first case, let S_i (respectively, T_i) be the set of all elements in S (respectively T) that come before s_i and A_i be the set of all elements of A that come before a_i (in the ordering $\prec$). When the algorithm updates the solution S, it places all of A after $S \setminus B$, removes all elements of B from S, and leaves all elements of $S \setminus B$ in the same relative order. Thus, $T_i \subseteq S_i$. It follows directly from the submodularity of f that

$$w_S(x) = \left\lfloor \frac{f(S_i + s_i) - f(S_i)}{\alpha} \right\rfloor \alpha \leq \left\lfloor \frac{f(T_i + s_i) - f(T_i)}{\alpha} \right\rfloor \alpha = w_T(x) \ .$$

Let $w_{(A,B)}$ be the weight function for k-exchange (A, B) and current solution S. We now show that $w_{(A,B)}(a_i) \leq w_T(a_i)$ for each element $a_i \in A$. In this case, we let A_i be the set of all elements of A that come before a_i (in the ordering $\prec$) and T_i be the set of all elements of T that come before a_i after applying (A, B) to S. When the algorithm updates the solution S, it places all elements of A after all of $S \setminus B$, removes all elements of B from S, and leaves all elements of A in the same relative order. Thus, $T_i \subseteq (S \setminus B) \cup A_i$ and so from the submodularity of f

$$w_{(A,B)}(a_i) = \left\lfloor \frac{f((S \setminus B) \cup A_i + a_i) - f((S \setminus B) \cup A_i)}{\alpha} \right\rfloor \alpha \leq \left\lfloor \frac{f(T_i + a_i) - f(T_i)}{\alpha} \right\rfloor \alpha = w_T(a_i) \ .$$

Finally, since Algorithm 2 applied (A, B) to S, we must have $w^2_{(A,B)}(A) > w^2_S(B)$, and since all weights are integer multiples of α, we must in fact have $w^2_{(A,B)}(A) \geq w^2_S(B) + \alpha^2$. From this inequality, together with the above bounds on w_S and $w_{(A,B)}$, we have

$$w^2_S(S) = \sum_{x \in S \setminus B} w_S(x)^2 + \sum_{x \in B} w_S(x)^2 \leq \sum_{x \in S \setminus B} w_S(x)^2 + \sum_{y \in A} w_{(A,B)}(y)^2 + \alpha^2$$
$$\leq \sum_{x \in S \setminus B} w_T(x)^2 + \sum_{y \in A} w_T(y)^2 + \alpha^2 = w^2_T(T) + \alpha^2 \ . \quad \blacktriangleleft$$

▶ **Theorem 7.** *For any value $\epsilon \in (0, 1)$, Algorithm 2 makes at most $O(n^3\epsilon^{-2})$ improvements.*

Proof. Because f is submodular, for any element e and any set $T \subseteq \mathcal{G}$, we have $f(T + e) - f(T) \leq f(\{e\}) \leq f(S_{init})$. In particular, for any solution $S \subseteq \mathcal{G}$ with associated weight function w, we have

$$w^2(S) = \sum_{e \in S} w(e)^2 \leq |S| f(S_{init})^2 \leq n f(S_{init})^2 \ .$$

Additionally, from Lemma 6, each improvement we apply must increase $w^2(S)$ by at least α^2, and hence the number of improvements that Algorithm 2 can make is at most

$$\frac{w^2(S) - f(S_{init})^2}{\alpha^2} \leq \frac{n f(S_{init})^2 - f(S_{init})^2}{\alpha^2}$$
$$= (n-1)\left(\frac{f(S_{init})}{\alpha}\right)^2 = (n-1)\frac{n^2}{\delta^2} = O(n^3\epsilon^{-2}) \ . \quad \blacktriangleleft$$

▶ **Corollary 8.** *For any $\epsilon > 0$, Algorithm 2 is a $\frac{k+3}{2} + \epsilon$ approximation algorithm, running in time $O(\epsilon^{-2}k^2n^{k^2+4})$.*

5 Open Questions

We do not currently have an example for which the locality gap of Algorithm 2 can be as bad as stated, even for specific k-exchange systems such as k-set packing. In the particular case of weighted independent set in $(k+1)$-claw free graphs, Berman [2] gives a tight example that shows his algorithm can return a set S with $\frac{k+1}{2}w(S) = w(O)$. His example uses only unit weights, and so the non-oblivious potential function is identical to the oblivious one. However, the algorithm of Feldman et al. (given here as Algorithm 1) considers a larger class of improvements than those considered by Berman, and so this example no longer applies, even in the linear case. For the unweighted variant, Hurkens and Schrijver [15] give a lower bound of $k/2 + \epsilon$, where ϵ depends on the size of the improvements considered. Because the non-oblivious local search routine performs the same as oblivious local search on instances with unit weights (since $1 = 1^2$), this lower bound applies to Algorithm 1 in the linear case. From a hardness perspective, the best known bound is the $\Omega(k/\ln k)$ NP-hardness result of Hazan, Safra, and Schwartz [14], for the special case of unweighted k-set packing.

Another interesting question is whether similar techniques can be adapted to apply to more general problems such as matroid k-parity in arbitrary matroids (here, even an improvement over k for the general linear case would be interesting) or to non-monotone submodular functions. A major difficulty with the latter generalization is our proof's dependence on the weights' non-negativity, as this assumption no longer holds if our approach is applied directly to non-monotone submodular functions.

Acknowledgments The author thanks Allan Borodin for providing comments on a preliminary version of this paper, as well as anonymous referees for providing further suggestions.

References

1. E. M. Arkin and R. Hassin. On local search for weighted k-set packing. In *Proc. of 5th ESA*, pages 13–22, 1997.
2. P. Berman. A d/2 approximation for maximum weight independent set in d-claw free graphs. *Nordic J. of Computing*, 7:178–184, Sept. 2000.
3. R. A. Brualdi. Induced matroids. *Proc. of the American Math. Soc.*, 29:213–221, 1971.
4. G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák. Maximizing a submodular set function subject to a matroid constraint. In *Proc. of 12th IPCO*, pages 182–196, 2007.
5. B. Chandra and M. Halldórsson. Greedy local improvement and weighted set packing approximation. In *Proc. of 10th ACM-SIAM SODA*, pages 169–176, 1999.
6. U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45:634–652, July 1998.
7. U. Feige, V. S. Mirrokni, and J. Vondrak. Maximizing non-monotone submodular functions. In *Proc. of 48th IEEE FOCS*, pages 461–471, 2007.
8. M. Feldman, J. S. Naor, and R. Schwartz. Nonmonotone submodular maximization via a structural continuous greedy algorithm. In *Proc. of 38th ICALP*, pages 342–353, 2011.
9. M. Feldman, J. S. Naor, R. Schwartz, and J. Ward. Improved approximations for k-exchange systems. In *Proc. of 19th ESA*, pages 784–798, 2011.
10. M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions—II. In *Polyhedral Combinatorics*, volume 8 of *Mathematical Programming Studies*, pages 73–87. Springer Berlin Heidelberg, 1978.
11. S. O. Gharan and J. Vondrák. Submodular maximization by simulated annealing. In *Proc. of 22nd ACM-SIAM SODA*, pages 1098–1116, 2011.
12. A. Gupta, A. Roth, G. Schoenebeck, and K. Talwar. Constrained non-monotone submodular maximization: Offline and secretary algorithms. In *Proc. of 7th WINE*, pages 246–257, 2010.
13. M. M. Halldórsson. Approximating discrete collections via local improvements. In *Proc. of 6th ACM-SIAM SODA*, pages 160–169, 1995.
14. E. Hazan, S. Safra, and O. Schwartz. On the complexity of approximating k-set packing. *Computational Complexity*, 15:20–39, May 2006.
15. C. A. J. 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. Discret. Math.*, 2(1):68–72, 1989.
16. S. Khanna, R. Motwani, M. Sudan, and U. Vazirani. On syntactic versus computational views of approximability. In *Proc. of 35th IEEE FOCS*, pages 819–830, 1994.
17. J. Lee, V. S. Mirrokni, V. Nagarajan, and M. Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In *Proc. of 41st ACM STOC*, pages 323–332, 2009.
18. J. Lee, M. Sviridenko, and J. Vondrák. Matroid matching: the power of local search. In *Proc. of 42nd ACM STOC*, pages 369–378, 2010.
19. J. Lee, M. Sviridenko, and J. Vondrák. Submodular maximization over multiple matroids via generalized exchange properties. *Math. of Oper. Res.*, 35(4):795 –806, Nov. 2010.
20. J. A. Soto. A simple PTAS for weighted matroid matching on strongly base orderable matroids. *Electronic Notes in Discrete Mathematics*, 37:75–80, Aug. 2011.

A Pumping Lemma for Pushdown Graphs of Any Level

Paweł Parys*

University of Warsaw, ul. Banacha 2, 02-097 Warszawa, Poland, parys@mimuw.edu.pl

Abstract

We present a pumping lemma for the class of ε-contractions of pushdown graphs of level n, for each n. A pumping lemma was proposed by Blumensath, but there is an irrecoverable error in his proof; we present a new proof. Our pumping lemma also improves the bounds given in the invalid paper of Blumensath.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases pushdown graph, ε-contraction, pumping lemma

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.54

1 Introduction

Higher-order pushdown systems are a very natural extension of pushdown systems. They were originally introduced by Maslov [10]. In a system of level n we have a level-n stack of level-$(n-1)$ stacks of ... of level-1 stacks. The idea is that the system 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. Higher-order pushdown systems have connections with several other concepts. A result of Knapik et al. [9] shows that higher-order pushdown systems generate the same trees as *safe* higher-order recursion schemes. Carayol and Wöhrle [2] proved that the ε-contractions of graphs generated by higher-order pushdown systems are exactly the graphs in the Caucal hierarchy [3]. Thus, all these graphs have decidable monadic second-order theories.

Even though higher-order pushdown systems generate important classes of graphs, useful characterizations of their structure are still rare. We still miss techniques for disproving membership in the pushdown hierarchy. In classical automata theory, pumping lemmas provide good tools for proving that a language cannot be defined by a finite automaton or by a pushdown automaton. For indexed languages, which are the languages recognized by pushdown systems of level 2, we have a pumping lemma of Hayashi [6], and a shrinking lemma of Gilman [4]. We also have a pumping lemma of Kartzow [7] for collapsible pushdown systems of level 2. On higher levels, similar results are still missing. Blumensath [1] published a pumping lemma for all levels of the higher-order pushdown hierarchy. Unfortunately, there is an irrecoverable error in his proof (cf. [11]).

Our main theorem is the following pumping lemma applicable to every level of the higher-order pushdown graph hierarchy.

▶ **Theorem 1.1.** *Let $\mathcal{A}$ be a pushdown system of level n, and L a regular language. Let G be the ε-contraction of the pushdown graph of $\mathcal{A}$; assume that it is finitely branching. Assume that in G there exists a path of length m from the initial configuration to some configuration*

* Work partially supported by the Polish Ministry of Science grant nr N N206 567840.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 54–65

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

c. Let $S_1 = (m+1) \cdot C_{\mathcal{A}L}$ *and* $S_j = 2^{S_{j-1}}$ *for* $2 \leq j \leq n$*, where* $C_{\mathcal{A}L}$ *is a constant which depends on* $\mathcal{A}$ *and on* L*. Assume also that in* G *there exists a path* p *of length at least* S_n*, which starts in* c *and belongs*[1] *to* L*. Then there are infinitely many paths in* G*, which start in* c*, belong to* L*, and end in configurations having the same state as the last configuration of* p*.*

This theorem is very similar to the pumping lemma proposed in [1]. Namely our Lemma 5.2 is an analogue of Corollary 16 from [1], and our Lemma 5.3 is an analogue of Theorem 61 from [1]; the above theorem (without the part about the regular language L) is obtained by composing these two lemmas.

Notice also that the bound S_n is $n-1$ times exponential in m, while the corresponding bound in [1] is $3n-1$ times exponential. Thus we obtain a better bound. Moreover, our bound is optimal, as explained in Section 6. The other difference is that our pumping preserves a regular property L of the paths, as well as the state of the last configuration.

2 Preliminaries

A *pushdown system* (PDS for short) of level n is given by a tuple $(A, \Gamma, \gamma_I, Q, q_I, \Delta, \lambda)$, where

- A is an input alphabet,
- Γ is a stack alphabet, and $\gamma_I \in \Gamma$ is an initial stack symbol,
- Q is a set of states, and $q_I \in Q$ is an initial state,
- $\Delta \subseteq Q \times \Gamma \times Q \times OP$ is a transition relation, where the set OP contains the operations pop^k and $\mathsf{push}^k(\alpha)$ for $1 \leq k \leq n$ and $\alpha \in \Gamma$,
- $\lambda\colon \Delta \to A \cup \{\varepsilon\}$ is a labelling of transitions.

In this paper, the letter n is always used for the level of the pushdown system.

For any alphabet Γ (of stack symbols) we define a *k-th level pushdown store* (k-pds for short) as an element of the following set Γ_*^k:

$$\Gamma_*^0 = \Gamma,$$
$$\Gamma_*^k = (\Gamma_*^{k-1})^* \qquad \text{for } 1 \leq k \leq n.$$

In other words, a 0-pds is just a single symbol, and a k-pds for $1 \leq k \leq n$ is a (possibly empty) sequence of $(k-1)$-pds's. The last element of a k-pds is also called the *topmost* one. For any $\alpha^k \in \Gamma_*^k$ and $\alpha^{k-1} \in \Gamma_*^{k-1}$ we write $\alpha^k : \alpha^{k-1}$ for the k-pds obtained from α^k by placing α^{k-1} at its end. The operator „:" is assumed to be right associative, i.e. $\alpha^2 : \alpha^1 : \alpha^0 = \alpha^2 : (\alpha^1 : \alpha^0)$. We say for $k \geq 1$ that a k-pds is *proper* if it is nonempty and every $(k-1)$-pds in it is proper; a 0-pds is always proper.

A *configuration* of $\mathcal{A}$ consists of a state and of a proper n-pds, i.e. it is an element of $Q \times \Gamma_*^n$ in which the n-pds is proper. The *initial* configuration consists of the initial state q_I and of the n-pds containing only one 0-pds, which is the initial stack symbol γ_I. For a configuration c, its state is denoted by $state(c)$, and its n-pds is denoted by $\pi(c)$.

Next, for configurations c, d we define when $c \vdash d$. Let α be the topmost 0-pds of $\pi(c)$. Assume that $(state(c), \alpha, state(d), op) \in \Delta$. We have two cases depending on op:

- if $op = \mathsf{pop}^k$ then $\pi(d)$ is obtained from $\pi(c)$ by replacing its topmost k-pds $\alpha^k : \alpha^{k-1}$ by α^k (i.e. we remove the topmost $(k-1)$-pds; in particular the topmost k-pds of $\pi(c)$ has to contain at least two $(k-1)$-pds's),

[1] Formally, the word consisting of labels on that path belongs to L.

- if $op = \mathsf{push}^k(\beta)$ then $\pi(d)$ is obtained from $\pi(c)$ by replacing its topmost k-pds $\alpha^k : \alpha^{k-1}$ by $(\alpha^k : \alpha^{k-1}) : \alpha^{k-1}$, and then by replacing its topmost 0-pds by β (i.e. we copy the topmost k-pds, and then we change the topmost symbol in the copy[2]).

A *run* is a function w from numbers $0, 1, \ldots, l$ (for some $l \geq 0$) to configurations such that $w(i-1) \vdash w(i)$ for $1 \leq i \leq l$. The number l is called the *length* of w, and denoted by $|w|$. We say that w is a run from $w(0)$ to $w(|w|)$. For $0 \leq x \leq y \leq |w|$ we can consider the *subrun of w from x to y*; this is the run of length $y - x$ which maps i to $w(i + x)$. For two runs v, w such that $v(|v|) = w(0)$ we can consider their *composition*; this is the run of length $|v| + |w|$ which maps $i \leq |v|$ to $v(i)$, and $i > |v|$ to $w(i - |v|)$. We say that a configuration d is *reachable* from a configuration c if there exists a run w from c to d.

The *pushdown graph of $\mathcal{A}$*, denoted by $PDG(\mathcal{A})$, is the directed graph consisting of configurations of $\mathcal{A}$ reachable from the initial configuration; there is an edge from a configuration c to a configuration d when $c \vdash d$. To each edge of $PDG(\mathcal{A})$ we can assign a label from $A \cup \{\varepsilon\}$ in the following way. Let c, d be configurations such that $c \vdash d$. Notice that the transition $\delta \in \Delta$ used between c and d (in the definition of $\vdash$) is uniquely determined. We label the edge from c to d by $\lambda(\delta)$. A run of $\mathcal{A}$ can also be interpreted as a path in $PDG(\mathcal{A})$, so it makes sense to talk about edges of a run, and about labels of these edges.

We define the *ε-contraction* of $PDG(\mathcal{A})$, denoted by $PDG(\mathcal{A})/\varepsilon$, which is a directed multigraph.[3] Its vertices are the initial configuration c_I, and configurations d such that there is a run from c_I to d in which the last edge is labelled by an element of A (i.e. not by ε). In $PDG(\mathcal{A})/\varepsilon$ there is an edge from c to d labelled by $a \in A$ when in $PDG(\mathcal{A})$ there is a path from c to d whose edges except the last one are labelled by ε, and the last edge is labelled by a. We say that $PDG(\mathcal{A})/\varepsilon$ is *finitely branching* if from each of its nodes there are only finitely many outgoing edges.

A *position* is a vector $x = (x_n, x_{n-1}, \ldots, x_1)$ of n positive integers. The symbol on position x in configuration c (which is an element of Γ) is defined as follows: we take the x_n-th (from the bottom) $(n-1)$-pds of $\pi(c)$, then its x_{n-1}-th $(n-2)$-pds, and so on. We say that x is a position of c, if at position x there is a symbol in c.

For $0 \leq k \leq n$, by $top^k(c)$ we denote the position of the bottommost symbol of the topmost k-pds of c. In particular $top^0(c)$ is the position of the topmost symbol in c.

For any run w, indices $0 \leq a \leq b \leq |w|$, and a position y of $w(b)$, we define a position $hist_w(b, y)(a)$. It is y when $b = a$. It is y also when $b = a + 1$, and the operation between $w(a)$ and $w(b)$ is pop^k, as well as when the operation is push^k and y is not in the topmost $(k-1)$-pds of $w(b)$. If the operation between $w(a)$ and $w(b)$ is push^k and y is in the topmost $(k-1)$-pds of $w(b)$, then $hist_w(b, y)(a)$ is the position of $w(a)$ from which a symbol was copied to y (i.e. this is y with the $(n-k+1)$-th coordinate decreased by 1). When $b > a+1$, $hist_w(b, y)(a)$ is defined (by induction) as $hist_w(a+1, hist_w(b, y)(a+1))(a)$. In other words, $hist_w(b, y)(a)$ is the (unique) position of $w(a)$, from which the symbol was copied to y in $w(b)$.

For $0 \leq k \leq n$, a run w, and an index $0 \leq b \leq |w|$ we define a set $pre^k_w(b)$ consisting of all indices a for which $0 \leq a \leq b$ and $hist_w(b, top^k(w(b)))(a) = top^k(w(a))$. Intuitively, $a \in pre^k_w(b)$ means that the topmost k-pds of $w(b)$ „comes from" the topmost k-pds of $w(a)$,

---

[2] In the classical definition the topmost symbol can be changed only when $k = 1$ (for $k \geq 2$ it has to be $\beta = \alpha$). Notice however that our theorems, true for every PDS, are in particular true for such restricted PDS's. On the other hand, it is not difficult to see that for any PDS $\mathcal{A}$ of level n there exists a PDS $\mathcal{B}$ of level n of this restricted form such that graphs $PDG(\mathcal{A})/\varepsilon$ and $PDG(\mathcal{B})/\varepsilon$ are isomorphic.

[3] In this graph, unlike in $PDG(\mathcal{A})$, we can have multiple edges between two nodes, each labeled by a different symbol.

in the sense that the topmost k-pds of $w(b)$ is a copy of the topmost k-pds of $w(a)$, but possibly some changes were done to it.

Example. Consider a PDS of level 3. Below, brackets are used in descriptions of pds's as follows: symbols taken in brackets form one 1-pds, 1-pds's taken in brackets form one 2-pds, and 2-pds's taken in brackets form one 3-pds. Consider a run w of length 6 in which $\pi(w(0)) = [[[ab]]]$ and the operations between consecutive configurations are:

$$\mathsf{push}^2(c),\ \mathsf{push}^3(d),\ \mathsf{pop}^1,\ \mathsf{push}^3(e),\ \mathsf{pop}^2,\ \mathsf{pop}^3.$$

The contents of the 3-pds's of the configurations in the run, and the *pre* sets, are presented in the table below.

i	$\pi(w(i))$	$pre^0_w(i)$	$pre^1_w(i)$	$pre^2_w(i)$	$pre^3_w(i)$
0	$[[[ab]]]$	$\{0\}$	$\{0\}$	$\{0\}$	$\{0\}$
1	$[[[ab][ac]]]$	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$	$\{0,1\}$
2	$[[[ab][ac]][[ab][ad]]]$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$	$\{0,1,2\}$
3	$[[[ab][ac]][[ab][a]]]$	$\{3\}$	$\{0,1,2,3\}$	$\{0,1,2,3\}$	$\{0,1,2,3\}$
4	$[[[ab][ac]][[ab][a]][[ab][e]]]$	$\{3,4\}$	$\{0,1,2,3,4\}$	$\{0,1,2,3,4\}$	$\{0,1,2,3,4\}$
5	$[[[ab][ac]][[ab][a]][[ab]]]$	$\{0,5\}$	$\{0,5\}$	$\{0,1,2,3,4,5\}$	$\{0,1,2,3,4,5\}$
6	$[[[ab][ac]][[ab][a]]]$	$\{3,6\}$	$\{0,1,2,3,6\}$	$\{0,1,2,3,6\}$	$\{0,1,2,3,4,5,6\}$

In configuration $w(0)$ symbol a is on position $(1,1,1)$ and symbol b is on position $(1,1,2)$. We have

$$hist_w(2,(2,2,1))(1) = (1,2,1) \qquad \text{and} \qquad hist_w(2,(2,2,1))(0) = (1,1,1).$$

Notice that positions y in $w(b)$ and $hist_w(b,y)(a)$ in $w(a)$ not necessarily contain the same symbol, for example on position $(1,2,2)$ in $w(1)$ we have c, and on position $(1,1,2)$ in $w(0)$ we have b, but $hist_w(1,(1,2,2))(0) = (1,1,2)$.

Easy properties. The following two propositions follow immediately from the definitions. These properties are often used implicitly later.

▶ **Proposition 2.1.** *Let w be a run, let $0 \le k \le n$, and let $0 \le a \le b \le c \le |w|$. Then*

- *$pre^k_w(b) \subseteq pre^{k+1}_w(b)$ (for $k < n$), and*
- *$a \in pre^k_w(b)$ and $b \in pre^k_w(c)$ implies $a \in pre^k_w(c)$, and*
- *$\{a,b\} \subseteq pre^k_w(c)$ implies $a \in pre^k_w(b)$.*

▶ **Proposition 2.2.** *Let w be a run, let $1 \le k \le n$, and let $0 \le a \le b \le |w|$ be such that $a \in pre^k_w(b)$. Then $a \in pre^{k-1}_w(b)$ if and only if, for all $a \le i \le b$, the size of the k-pds of $w(i)$ containing $hist_w(b, top^k(w(b)))(i)$ is not smaller than the size of the topmost k-pds of $w(a)$.*

3 Types of configurations

Let $\mathcal{A} = (A, \Gamma, \gamma_I, Q, q_I, \Delta, \lambda)$ be a PDS of level n. Below we define a function $type_{\mathcal{A}}$ which assigns to every configuration of $\mathcal{A}$ an element of a finite set $\mathcal{T}_{\mathcal{A}}$. The important properties of the $type_{\mathcal{A}}$ function are listed below, in the three facts.

▶ **Fact 3.1.** *Let $\mathcal{A}$ be a PDS of level n. Let w be a run of $\mathcal{A}$ such that $0 \in pre^0_w(|w|)$, and let c be a configuration such that $type_{\mathcal{A}}(w(0)) = type_{\mathcal{A}}(c)$. Then there exists a run v from c such that*

1. *if* $\pi(w(0)) \neq \pi(w(|w|))$ *then* $\pi(v(0)) \neq \pi(v(|v|))$, *and*
2. $0 \in pre_v^0(|v|)$, *and*
3. *all edges of* w *are labelled by* ε *if and only if all edges of* v *are labelled by* ε, *and*
4. $type_{\mathcal{A}}(w(|w|)) = type_{\mathcal{A}}(v(|v|))$.

▶ **Fact 3.2.** *Let* $\mathcal{A}$ *be a PDS of level* n. *Let* w *be a run of* $\mathcal{A}$ *such that at least one of its edges is not labelled by* ε, *and the position* $top^0(w(0))$ *is present in every configuration of* w. *Let* c *be a configuration such that* $type_{\mathcal{A}}(w(0)) = type_{\mathcal{A}}(c)$. *Then there exists a run* v *from* c *such that at least one of its edges is not labelled by* ε, *and the position* $top^0(c)$ *is present in every configuration of* v.

▶ **Fact 3.3.** *Let* $\mathcal{A}$ *be a PDS of level* n. *Let* w *be a run of* $\mathcal{A}$, *and let* c *be a configuration such that* $type_{\mathcal{A}}(w(0)) = type_{\mathcal{A}}(c)$. *Then there exists a run* v *from* c *such that* $state(v(|v|)) = state(w(|w|))$.

Before we define types of configurations, we define types of k-pds's, for each k. The main idea is that we have to characterize special kind of runs, called k-returns, as well as runs as described by Facts 3.2 and 3.3.

▶ **Definition 3.4.** Let $1 \leq r \leq n$, and let w be a run. We say that w is an *r-return* if

- the topmost r-pds of $w(0)$ contains at least two $(r-1)$-pds's, and
- $hist_w(|w|, top^{r-1}(w(|w|)))(0)$ is the bottommost position of the $(r-1)$-pds just below the topmost $(r-1)$-pds of $w(0)$, and
- $pre_w^{r-1}(|w|) = \{|w|\}$.

In other words, w is an r-return when the topmost r-pds of $w(|w|)$ is obtained from the topmost r-pds of $w(0)$ by removing its topmost $(r-1)$-pds (but not only in the sense of contents, but we require that really it was obtained this way). In particular we have the following proposition.

▶ **Proposition 3.5.** *Let* w *be an* r*-return. Then the topmost* r*-pds of* $w(0)$ *after removing its topmost* $(r-1)$*-pds is equal to the topmost* r*-pds of* $w(|w|)$.

Example. Consider a PDS of level 2, and a run w of length 6 in which $\pi(w(0)) = [[ab][cd]]$, and the operations between consecutive configurations are:

$$\mathsf{push}^2(e),\ \mathsf{pop}^1,\ \mathsf{pop}^2,\ \mathsf{pop}^1,\ \mathsf{push}^1(d),\ \mathsf{pop}^1.$$

The contents of the 2-pds's of the configurations in the run are presented in the table below.

i	0	1	2	3	4	5	6
$\pi(w(i))$	$[[ab][cd]]$	$[[ab][cd][ce]]$	$[[ab][cd][c]]$	$[[ab][cd]]$	$[[ab][c]]$	$[[ab][cd]]$	$[[ab][c]]$

The subruns of w from 0 to 2, from 0 to 4, from 1 to 2, from 3 to 4, and from 5 to 6 are 1-returns; the subruns of w from 1 to 3, and from 2 to 3 are 2-returns. These are the only subruns of w being returns, in particular w is not a 1-return because $4 \in pre_w^0(6)$.

We are going to define a type of a k-pds for each $0 \leq k \leq n$. A set of possible level-k types (types of k-pds's) will be denoted by $\mathcal{T}^k$. We also define a set $\mathcal{D}^k$; its elements correspond to kinds of runs (this correspondence is formalized in the "agrees with" notion).

▶ **Definition 3.6.** We define $\mathcal{T}^k$ (where $0 \leq k \leq n$) by induction on k, going down from $k = n$ to $k = 0$. Let $0 \leq k \leq n$. Assume we have already defined sets $\mathcal{T}^i$ for $k+1 \leq i \leq n$. We take

$$\mathcal{D}^k = Q \cup \bigcup_{r=k+1}^{n} \{r\} \times \Big(\{\text{non-}\varepsilon\} \cup \Big(\{0,1\} \times \mathcal{P}(\mathcal{T}^n) \times \mathcal{P}(\mathcal{T}^{n-1}) \times \cdots \times \mathcal{P}(\mathcal{T}^{r+1}) \times Q \times \{0,1\}\Big)\Big),$$

$$\mathcal{T}^k = \mathcal{P}(\mathcal{T}^n) \times \mathcal{P}(\mathcal{T}^{n-1}) \times \cdots \times \mathcal{P}(\mathcal{T}^{k+1}) \times Q \times \mathcal{D}^k,$$

where by $\mathcal{P}(X)$ we denote the power set of X (the set of all subsets of X).

▶ **Definition 3.7.** We define $type(\alpha^k) \subseteq \mathcal{T}^k$ for a k-pds α^k (where $0 \leq k \leq n$) by induction on k, going down from $k = n$ to $k = 0$. Let $0 \leq k \leq n$. Assume we have already defined sets *type* for i-pds's for $k+1 \leq i \leq n$.

1. Let $t = (r, f, \xi^n, \xi^{n-1}, \ldots, \xi^{r+1}, q, g) \in \mathcal{D}^k$, and let w be a run. Decompose $\pi(w(|w|)) = \beta^n : \beta^{n-1} : \cdots : \beta^r$. We say that w *agrees with* t if
   - w is an r-return, and
   - each edge of w is labelled by ε if and only if $f = 0$, and
   - $type(\beta^i) = \xi^i$ for $r+1 \leq i \leq n$, and
   - $q = state(w(|w|))$, and
   - $\pi(w(|w|))$ can be obtained from $\pi(w(0))$ by removing its topmost $(r-1)$-pds if and only if $g = 0$.
2. We say that a run w *agrees with* $(r, \text{non-}\varepsilon) \in \mathcal{D}^k$ if at least one edge of w is labelled by an element of A, and position $top^{r-1}(w(0))$ is present in every configuration of w.
3. We say that a run w *agrees with* $q \in \mathcal{D}^k \cap Q$ if $state(w(|w|)) = q$.
4. Let $t = (\rho^n, \rho^{n-1}, \ldots, \rho^{k+1}, p, t') \in \mathcal{T}^k$, and let α^k be a k-pds. We say that $t \in type(\alpha^k)$ if the following is true.

   For $k+1 \leq i \leq n$, let α^i be an i-pds such that $type(\alpha^i) = \rho^i$. Then there exists a run from $(p, \alpha^n : \alpha^{n-1} : \cdots : \alpha^k)$ which agrees with t'.

In point 4 of the above definition we mean that for all appropriate $\alpha^{k+1}, \alpha^{k+2}, \ldots, \alpha^n$ the run exists (and not that there exist appropriate $\alpha^{k+1}, \alpha^{k+2}, \ldots, \alpha^n$ such that the run exists). However in fact the „there exists" variant would be equivalent; this is described by the following lemma, which is the main technical result about types.

▶ **Lemma 3.8.** *Let $0 \leq k \leq n$, let $t \in \mathcal{D}^k$, and let w be a run which agrees with t. Decompose $\pi(w(0)) = \alpha^n : \alpha^{n-1} : \cdots : \alpha^k$. Then*

$$(type(\alpha^n), type(\alpha^{n-1}), \ldots, type(\alpha^{k+1}), state(w(0)), t) \in type(\alpha^k).$$

The proof of this lemma is tedious but rather straightforward. Finally, we define types of configurations.

▶ **Definition 3.9.** Let $\mathcal{T}_{\mathcal{A}} = \mathcal{P}(\mathcal{T}^n) \times \mathcal{P}(\mathcal{T}^{n-1}) \times \cdots \times \mathcal{P}(\mathcal{T}^1) \times \Gamma \times Q$. For a configuration $c = (q, \alpha^n : \alpha^{n-1} : \cdots : \alpha^0)$, let

$$type_{\mathcal{A}}(c) = (type(\alpha^n), type(\alpha^{n-1}), \ldots, type(\alpha^1), \alpha^0, q).$$

Using Lemma 3.8 it is not difficult to show that Facts 3.1–3.3 for such definition of a type.

4 Pumping of pushdown graphs

The following technical lemma describes how pushdown graphs can be pumped.

▶ **Lemma 4.1.** *Let $\mathcal{A}$ be a PDS of level n, let $0 \leq k \leq n$, let w be a run of $\mathcal{A}$, and let $G \subseteq pre^k_w(|w|) - \{|w|\}$. Let α^k be the k-pds of $w(0)$ containing $hist_w(|w|, top^k(w(|w|)))(0)$. For $1 \leq j \leq k$, let r_j be the maximum of the sizes of the j-pds's in α^k. Define*

$$N_0 = |\mathcal{T}_\mathcal{A}| + 1 \qquad \text{and} \qquad N_j = r_j \cdot 2^{N_{j-1}} \qquad \text{for } 1 \leq j \leq k.$$

Assume that $|G| \geq N_k$. Then there exist indices $0 \leq x < y < z \leq |w|$ such that

1. *$type_\mathcal{A}(w(x)) = type_\mathcal{A}(w(y))$, and*
2. *$x \in pre^0_w(y)$ and $y \in pre^k_w(|w|)$, and*
3. *either $\pi(w(x)) \neq \pi(w(y))$, or $G \cap \{x, x+1, \ldots, y-1\} \neq \emptyset$, and*
4. *$z - 1 \in G$ and $top^0(w(y))$ is present in every configuration of the subrun of w from y to z.*

Let us comment on the statement of this lemma. The essence of the lemma is that in every appropriately long run one can find indices x, y such that $type_\mathcal{A}(w(x)) = type_\mathcal{A}(w(y))$ and $x \in pre^0_w(y)$. Notice that the notion "appropriately long" depends on the size of the stack in $w(0)$: when one starts from a bigger stack, we require a longer run. Then Fact 3.1 can be applied to the fragment of w between x and y, so this fragment can be pumped (repeated forever). The lemma is more complicated for technical reasons. The problem is that pumping any fragment of a run is not interesting enough. For example the fragment between x and y can be a loop doing nothing; we are not satisfied with finding such a loop. For this reason we have introduced the set G of "good" indices, and we assume that this set is big enough. Our goal is to have some element of G in the fragment between x and y (the second variant of condition 3). However this is not always possible, and we sometimes get the first variant of condition 3; the intuition is that then we can show (using also index z) that the graph has to be infinitely branching.

The above lemma is proved by induction on k. For $k = 0$ we have $|G| \geq |\mathcal{T}_\mathcal{A}| + 1$ and there are only $|\mathcal{T}_\mathcal{A}|$ possible values of $type_\mathcal{A}$, so there exist two indices $x, y \in G$ such that $x < y$ and $type_\mathcal{A}(w(x)) = type_\mathcal{A}(w(y))$ (we get condition 1). By assumption we know that $x, y \in pre^0_w(|w|)$; this implies that $x \in pre^0_w(y)$ (we get condition 2). We have condition 3 because $x \in G$. We take $z = y + 1$. We have $z - 1 \in G$. Because $y \in pre^0_w(|w|)$, position $top^0(w(y))$ is present in $w(z)$ (we get condition 4).

For $k > 0$ we make the induction step using the following lemma about sequences of integers. For $0 \leq i \leq |w|$ as a_i we take the size of the k-pds of $w(i)$ containing $hist_w(|w|, top^k(w(|w|)))(i)$.

▶ **Lemma 4.2.** *Let $N \geq 1$ be a natural number, let $a_0, a_1, \ldots, a_M$ be a sequence of positive integers such that $|a_i - a_{i-1}| \leq 1$ for $1 \leq i \leq M$. Let $G \subseteq \{0, 1, \ldots, M-1\}$ be such that $|G| \geq a_0 \cdot 2^N$. Then there exist two indices b, e such that $0 \leq b < e \leq M$ and $e - 1 \in G$, and*

1. *for each i such that $b \leq i \leq e$ we have $a_i \geq a_b$, and*
2. *for each i such that $0 \leq i \leq b - 1$ we have $a_i \geq a_b + 1$, and*
3. *$|H_{b,e}| \geq N$, where*

$$H_{b,e} = \{i \colon b \leq i \leq e-1 \wedge \forall_j (i \leq j \leq e \Rightarrow a_j \geq a_i) \wedge \\ \wedge \exists_{g \in G} (g \geq i \wedge \forall_j (i+1 \leq j \leq g \Rightarrow a_j \geq a_i + 1))\}.$$

5 Finitely branching ε-contractions of pushdown graphs

In this section we show how finitely branching ε-contractions of pushdown graphs can be pumped; we prove Theorem 1.1. First we give an auxiliary lemma, which describes how the assumption about finite branching can be used. Then we have two lemmas, which are then composed together into Theorem 1.1. Lemma 5.2 tells us that a short run from the initial configuration cannot finish in a configuration having a big stack. Lemma 5.3 is similar to Theorem 1.1, but instead of assuming that a configuration can be reached with a short run from the initial configuration, we assume that its stack is small (and this assumption will be then satisfied thanks to Lemma 5.2).

▶ **Lemma 5.1.** *Let $\mathcal{A}$ be a PDS of level n, let w be a run of $\mathcal{A}$ such that $w(0)$ is reachable from the initial configuration, and let $0 \leq x < y \leq |w|-1$ be indices such that $type_{\mathcal{A}}(w(x)) = type_{\mathcal{A}}(w(y))$, and $x \in pre^0_w(y)$, and $\pi(w(x)) \neq \pi(w(y))$. Assume that $top^0(y)$ is present in every configuration of the subrun of w from y to $|w|$. Assume also that every edge of w between x and y is labelled by ε, and at least one edge of w between y and $|w|$ is not labelled by ε. Then $PDG(\mathcal{A})/\varepsilon$ is not finitely branching.*

Proof. Without loss of generality, we assume that w begins in the initial configuration; we can obtain such a situation by appending before w any run from the initial configuration to $w(0)$, and appropriately shifting x and y. Let g be the smallest index ($0 \leq g \leq x$) such that every edge between g and x is labelled by ε. Then $w(g)$ is a node of $PDS(\mathcal{A})/\varepsilon$.

We want to create a sequence of runs $v_1, v_2, v_3, \ldots$ such that for each $i \geq 1$ we have

a) $v_1(0) = w(x)$ and $v_i(0) = v_{i-1}(|v_{i-1}|)$ for $i > 1$, and
b) $\pi(v_i(0)) \neq \pi(v_i(|v_i|))$, and
c) $0 \in pre^0_{v_i}(|v_i|)$, and
d) every edge of v_i is labelled by ε, and
e) $type_{\mathcal{A}}(v_i(0)) = type_{\mathcal{A}}(v_i(|v_i|))$.

As v_1 we can take the subrun of w from x to y. Assume that we already have v_i for some $i \geq 1$. We use Fact 3.1 for v_i (as w) and $v_i(|v_i|)$ (as c); thanks to properties c) and e) its assumptions are satisfied. We obtain a run v_{i+1} from $v_i(|v_i|)$. Conditions 1–4 of the fact immediately give us conditions b–e for v_{i+1}.

Notice, for each $i \geq 1$, that because $0 \in pre^0_{v_i}(|v_i|)$ and $\pi(v_i(0)) \neq \pi(v_i(|v_i|))$, position $top^0(v_i(|v_i|))$ (which is $top^0(v_{i+1}(0))$) is lexicographically greater than $top^0(v_i(0))$. Thus every $top^0(v_i(0))$ is different.

For every $i \geq 1$ we do the following. From condition e) and from $type_{\mathcal{A}}(w(x)) = type_{\mathcal{A}}(w(y))$ we know that $type_{\mathcal{A}}(v_i(0)) = type_{\mathcal{A}}(w(y))$. We use Fact 3.2 for the subrun of w from y to $|w|$ (as w), and for $v_i(0)$ (as c). We obtain a run u_i from $v_i(0)$ such that at least one of its edges is not labelled by ε, and position $top^0(v_i(0))$ is present in every configuration of u_i. We can assume that only the last edge of u_i is not labelled by ε (we obtain this situation by cutting u_i after the first edge not labelled by ε). Now compose the subrun of w from g to x, runs $v_1, v_2, \ldots, v_{i-1}$, and run u_i. We obtain a run from $w(g)$ such that only its last edge is not labelled by ε. Thus $u_i(|u_i|)$ is a successor of $w(g)$ in $PDG(\mathcal{A})/\varepsilon$, in which position $top^0(v_i(0))$ is present. As each position $top^0(v_i(0))$ is different, they cannot be all present in only finitely many configurations, so among $u_i(|u_i|)$ there are infinitely many different configurations. This means that $PDG(\mathcal{A})/\varepsilon$ is not finitely branching. ◀

▶ **Lemma 5.2.** *Let $\mathcal{A}$ be a PDS of level n such that $PDG(\mathcal{A})/\varepsilon$ is finitely branching. Let w be a run which begins in the initial configuration, and whose last edge is not labelled by ε.*

Let m be the number of edges of w not labelled by ε. Let

$$M_1 = (m+1)\cdot(|\mathcal{T}_\mathcal{A}|+1) \qquad \text{and} \qquad M_j = 2^{M_{j-1}} \qquad \text{for } 2 \le j \le n.$$

Then, for $1 \le k \le n$, the size of any k-pds of $w(|w|)$ is at most M_k.

Proof. Induction on m. Notice that $m \ge 1$. Define

$$M'_1 = m\cdot(|\mathcal{T}_\mathcal{A}|+1) \qquad \text{and} \qquad M'_j = 2^{M'_{j-1}} \qquad \text{for } 2 \le j \le n.$$

Let b be the index such that the $(m-1)$-st edge of w not labelled by ε is between $w(b-1)$ and $w(b)$; if $m = 1$ we take $b = 0$. From the induction assumption, used for the subrun of w from 0 to b, we know, for $1 \le k \le n$, that the size of any k-pds of $w(b)$ is at most M'_k. This is also true for $m = 1$, as $M'_k \ge 1$.

Assume that for some k ($1 \le k \le n$) the size of some k-pds of $w(|w|)$ is greater than M_k. Let s be the bottommost position of such a k-pds. Let v be the subrun of w from b to $|w|$. For $0 \le i \le |v|$, let a_i be the size of the k-pds of $v(i)$ containing $hist_v(|v|,s)(i)$. We have $a_{|v|} \ge M_k$ and $a_0 \le M'_k$. Of course $|a_{i-1} - a_i| \le 1$ for $1 \le i \le |v|$. Let

$$G = \{i\colon 0 \le i \le |v|-1 \wedge \forall_j (i+1 \le j \le |v| \Rightarrow a_j \ge a_i + 1)\}.$$

Notice that $|G| \ge M_k - M'_k$, as for each j such that $M'_k \le j \le M_k - 1$ in G we have the last index i such that $a_i = j$. Let e be the greatest index such that $e-1 \in G$; let v' be the subrun of v from 0 to e. Define

$$N_0 = |\mathcal{T}_\mathcal{A}| + 1 \qquad \text{and} \qquad N_i = M'_i \cdot 2^{N_{i-1}} \qquad \text{for } 1 \le i \le k-1.$$

We are going to use Lemma 4.1 for $k-1$ (as k), for the run v' (as w), and for G. We have to check that its assumptions are satisfied. We need to check that $G \subseteq pre_v^{k-1}(e)$. Because only the topmost k-pds can change its size, and $a_i \ne a_{i+1}$ for $i \in G$, it follows that $hist_v(|v|,s)(i) = top^k(v(i))$ for $i \in G \cup \{e\}$, which means that $G \subseteq pre_v^k(e)$. As additionally $a_j \ge a_i$ for $i \in G$, $i \le j \le |v|$, from Proposition 2.2 we get $G \subseteq pre_v^{k-1}(e)$, as required. We also need to check that G has enough elements; this is a straightforward calculation.

From Lemma 4.1 we obtain indices $0 \le x < y < z \le e$ such that

1. $type_\mathcal{A}(v(x)) = type_\mathcal{A}(v(y))$, and
2. $x \in pre_v^0(y)$, and
3. either $\pi(v(x)) \ne \pi(v(y))$, or $G \cap \{x, x+1, \ldots, y-1\} \ne \emptyset$, and
4. $z-1 \in G$ and $top^0(v(y))$ is present in every configuration of the subrun of v from y to z.

Is it possible that $\pi(v(x)) = \pi(v(y))$? As additionally $x \in pre_v^0(y)$ (condition 2), this would mean that for every position p in $v(y)$ we have $hist_v(y,p)(x) = p$ (between $v(x)$ and $v(y)$ some new fragments of the n-pds were added and then removed; it is impossible that we have first removed something and then reproduced it). In particular a_x and a_y describe the size of the same k-pds, so $a_x = a_y$. Moreover $a_i \ge a_x$ for $x \le i \le y$. But condition 3 implies that there is some $g \in G \cap \{x, x+1, \ldots, y-1\}$. This is impossible, as we have $a_y \ge a_g + 1$ (by definition of G), and $a_g \ge a_x$, which means that $a_x \ne a_y$. Thus we always have $\pi(v(x)) \ne \pi(v(y))$.

Because $z-1 \in G$, we have $a_{z-1} \ne a_z$, so since only the topmost k-pds can change its size, we know that $hist_v(|v|,s)(z) = top^k(v(z))$. Additionally $a_i \ge a_z = a_{z-1} + 1$ for $z \le i \le |v|$ (by definition of G), which means that $top^{k-1}(v(z))$ is present in every configuration of the subrun of v from z to $|v|$. Since $top^0(v(y))$ is present in $v(z-1)$ (condition 4), we know that $top^0(v(y))$ is (lexicographically) below $top^{k-1}(v(z))$, so one cannot remove $top^0(v(y))$

without removing $top^{k-1}(v(z))$. It follows that $top^0(v(y))$ is present in every configuration of the subrun of v from y to $|v|$.

Recall also that the last edge of v is not labelled by ε, and all earlier edges are labelled by ε. So every edge of v between x and y is labelled by ε, and at least one edge of v between y and $|v|$ is not labelled by ε. Thus the assumptions of Lemma 5.1 (where v is taken as w) are satisfied. We get that $PDG(\mathcal{A})/\varepsilon$ is not finitely branching, which contradicts with our assumption. ◀

▶ **Lemma 5.3.** *Let $\mathcal{A}$ be a PDS of level n such that $PDG(\mathcal{A})/\varepsilon$ is finitely branching, and let w be a run of $\mathcal{A}$ such that $w(0)$ is reachable from the initial configuration. For $1 \le j \le n$, let r_j be the maximum of the sizes of j-pds's of $w(0)$. Define*

$$N_0 = |\mathcal{T}_{\mathcal{A}}| + 1 \qquad \text{and} \qquad N_j = r_j \cdot 2^{N_{j-1}} \qquad \text{for } 1 \le j \le n.$$

Assume that at least N_n edges of w are not labelled by ε. Then for each $j \in \mathbb{N}$ there exist a run w_j from $w(0)$ which has at least j edges not labelled by ε, and such that $state(w_j(|w_j|)) = state(w(|w|))$.

Proof. Let G be the set of indices i ($0 \le i \le |w| - 1$) such that the edge between $w(i)$ and $w(i+1)$ is not labelled by ε. We use Lemma 4.1 for n (as k), for run w, and set G. Of course $G \subseteq pre^n_w(|w|)$, as $pre^n_w(|w|)$ by definition contains all numbers from 0 to $|w|$. We also have $|G| \ge N_n$, which is the required size. From the lemma we obtain indices $0 \le x < y < z \le |w|$ such that

1. $type_{\mathcal{A}}(w(x)) = type_{\mathcal{A}}(w(y))$, and
2. $x \in pre^0_w(y)$, and
3. either $\pi(w(x)) \ne \pi(w(y))$, or $G \cap \{x, x+1, \dots, y-1\} \ne \emptyset$, and
4. $z - 1 \in G$ and $top^0(w(y))$ is present in every configuration of the subrun of w from y to z.

Assume first that every edge of w between x and y is labelled by ε. By condition 3 we see that $\pi(w(x)) \ne \pi(w(y))$. Notice also that at least one edge of w between y and z is not labelled by ε, namely the last edge (as $z - 1 \in G$). The assumptions of Lemma 5.1 are satisfied; we get that $PDG(\mathcal{A})/\varepsilon$ is not finitely branching, which contradicts with our assumption. Thus at least one edge of w between x and y is not labelled by ε.

We want to create a sequence of runs $v_1, v_2, v_3, \dots$ beginning at $w(x)$ such that for each $j \ge 1$ we have

a) $0 \in pre^0_{v_j}(|v_j|)$, and
b) at least j edges of v_j are not labelled by ε, and
c) $type_{\mathcal{A}}(v_j(0)) = type_{\mathcal{A}}(v_j(|v_j|))$.

As v_1 we can take the subrun of w from x to y. Assume that we already have v_j for some $j \ge 1$. We use Fact 3.1 for v_j (as w) and $v_j(|v_j|)$ (as c); thanks to properties a) and c) its assumptions are satisfied. We obtain a run v from $v_j(|v_j|)$. Let v_{j+1} be the composition of runs v_j and v. Condition 2 of the fact says that $0 \in pre^0_v(|v|)$; together with $0 \in pre^0_{v_j}(|v_j|)$ it gives us that $0 \in pre^0_{v_{j+1}}(|v_{j+1}|)$. Condition 3 of the fact says that at least one edge of v is not labelled by ε; thus at least $j+1$ edges of v_{j+1} are not labelled by ε. Condition 4 of the fact says that $type_{\mathcal{A}}(v(0)) = type_{\mathcal{A}}(v(|v|))$; thus $type_{\mathcal{A}}(v_{j+1}(0)) = type_{\mathcal{A}}(v_{j+1}(|v_{j+1}|))$.

Next, we use Fact 3.3 for the subrun of w from y to $|w|$ and for $v_j(|v_j|)$; we obtain a run v'_j from $v_j(|v_j|)$ such that $state(v'_j(|v'_j|)) = state(w(|w|))$. Finally, as w_j we take the composition of the subrun of w from 0 to x with run v_i and with run v'_i; this run satisfies the thesis of the lemma. ◀

Proof (Theorem 1.1). First we consider the following special case. Assume that the language L contains all words. Assume also that the set of states of $\mathcal{A}$ is of the form $Q \times \{0,1\}$, and a transition is labelled by ε if and only if it leads to a state with 0 on the second coordinate. Then we take $C_{\mathcal{A}L} = 3 \cdot (|\mathcal{T}_\mathcal{A}| + 1) \cdot 2^{|\mathcal{T}_\mathcal{A}|+1}$. Because in $PDG(\mathcal{A})/\varepsilon$ we have a path of length m from the initial configuration to c, there exists a run w from the initial configuration to c such that exactly m of its edges are not labelled by ε, in particular the last one. Let

$$M_1 = (m+1) \cdot (|\mathcal{T}_\mathcal{A}| + 1) \qquad \text{and} \qquad M_j = 2^{M_{j-1}} \qquad \text{for } 2 \leq j \leq n.$$

By Lemma 5.2 we know, for $1 \leq k \leq n$, that the size of any k-pds of c is at most M_k. Let

$$N_0 = |\mathcal{T}_\mathcal{A}| + 1 \qquad \text{and} \qquad N_j = M_j \cdot 2^{N_{j-1}} \qquad \text{for } 1 \leq j \leq n.$$

A straightforward calculation proves that $S_n \geq N_n$. Because in $PDG(\mathcal{A})/\varepsilon$ we have a path of length S_n starting at c, there exists a run v starting at c such that at least $S_n \geq N_n$ of its edges are not labelled by ε. We use Lemma 5.3 for the run v (as w). It says that there exist runs w_j from c having arbitrarily many edges not labelled by ε, and such that $w_j(|w_j|)$ and $w(|w|)$ have the same state. Since one state is reached either only by ε-transitions or only by non-ε-transitions, the last edge of w_j is not labelled by ε, because the last edge of w was not labelled by ε. It means that there are arbitrarily many paths in $PDG(\mathcal{A})/\varepsilon$ starting at c, and ending in configurations with state $state(w(|w|))$.

Next, consider a situation where $\mathcal{A}$ is arbitrary, but L still contains all words. Then we convert $\mathcal{A}$ to $\mathcal{A}'$ having the above form. We simply product the states Q of $\mathcal{A}$ by $\{0,1\}$; for every transition $\delta = (q_1, \gamma, q_2, op)$ of $\mathcal{A}$, in $\mathcal{A}'$ we have, for $i = 0, 1$, transitions $((q_1, i), \gamma, (q_2, 0), op)$ if $\lambda(\delta) = \varepsilon$, or $((q_1, i), \gamma, (q_2, 1), op)$ otherwise. The initial state gets 1 on the second coordinate. Notice that only configurations having 1 on the second coordinate are in $PDG(\mathcal{A}')/\varepsilon$. Moreover there is an edge between two configurations in $PDG(\mathcal{A})/\varepsilon$ if and only if there is an edge between corresponding (obtained by putting 1 on the second coordinate of the state) configurations in $PDG(\mathcal{A}')/\varepsilon$. So the two graphs are isomorphic, thus the theorem for one of them immediately implies the theorem for the other.

For an arbitrary language L and arbitrary PDS $\mathcal{A}$ the theorem is true, because we can make a product of $\mathcal{A}$ with a finite automaton recognizing L. ◀

6 Example application

Let $\varphi\colon \mathbb{N} \to \mathbb{N}$ be an unbounded function. Let $f_1^\varphi(x) = x \cdot \varphi(x)$ and $f_{k+1}^\varphi(x) = 2^{f_k^\varphi(x)}$ for $k \geq 1$. Consider the tree T_n^φ whose nodes are

$$\{0^i 1^j : i \geq 0, j \leq f_n^\varphi(i+2) + 1\},$$

and a node w is connected with a node wa by an edge labelled by a (where w is a word and $a \in \{0,1\}$ is a letter). This tree is not isomorphic to the ε-contraction of any pushdown graph of level n.

Heading for a contradiction, assume that T_n^φ is isomorphic to $PDG(\mathcal{A})/\varepsilon$ for some pushdown system $\mathcal{A}$ of level n. In this isomorphism, the empty word in T_n^φ has to correspond to the initial configuration (as it is the only configuration which can have no predecessors). Choose $i \in \mathbb{N}$ such that $\varphi(i+2) \geq C_{\mathcal{A}L}$ (where $C_{\mathcal{A}L}$ is the constant from Theorem 1.1, for $L = \{0,1\}^*$). Let c be the configuration corresponding to $0^i 1$, and d the configuration corresponding to $0^i 1^{f_n^\varphi(i+2)+1}$. We use Theorem 1.1 for the path from the initial configuration

to c and for the path from c to d; their length is, respectively, $i+1$ and $f_n^\varphi(i+2)$ (which is greater or equal to S_n from the theorem). Thus we obtain infinitely many paths starting in 0^i1, which contradicts the definition of T_n^φ.

On the other hand it is known that when the function φ is constant, then tree T_n^φ is isomorphic to $PDG(\mathcal{A})/\varepsilon$ for some pushdown system $\mathcal{A}$. See e.g. [1], Example 9, where a very similar pushdown system is constructed. In this sense the length required in Theorem 1.1 is the smallest possible: S_n has to be $n-1$ times exponential in m.

7 Future work

As a continuation of this work, we have recently [8] generalized Theorem 1.1 to *collapsible* pushdown systems. Collapsible pushdown systems are an extension of higher-order pushdown systems, in which an additional operation, called *collapse*, can be performed. Trees generated by these systems correspond to all higher-order recursion schemes [5], not only to safe ones.

Our pumping lemma talks only about the length of paths, and about a regular condition on the labels on them, hence its applications are rather limited. It would be useful to show a pumping lemma which describes more precisely how the new paths (as sequences of labels) can be constructed from the original paths, similarly to the classical pumping lemma for finite automata or pushdown automata.

Acknowledgement. I would like to thank Alexander Kartzow for many useful comments.

References

1 Achim Blumensath. On the structure of graphs in the caucal hierarchy. *Theor. Comput. Sci.*, 400(1-3):19–45, 2008.

2 Arnaud Carayol and Stefan Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS*, pages 112–123. Springer, 2003.

3 Didier Caucal. On infinite terms having a decidable monadic theory. In Krzysztof Diks and Wojciech Rytter, editors, *MFCS*, volume 2420 of *Lecture Notes in Computer Science*, pages 165–176. Springer, 2002.

4 Robert H. Gilman. A shrinking lemma for indexed languages. *Theor. Comput. Sci.*, 163(1&2):277–281, 1996.

5 Matthew Hague, Andrzej S. Murawski, C.-H. Luke Ong, and Olivier Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, pages 452–461. IEEE Computer Society, 2008.

6 Takeshi Hayashi. On derivation trees of indexed grammars. *Publ. RIMS, Kyoto Univ.*, 9:61–92, 1973.

7 Alexander Kartzow. A pumping lemma for collapsible pushdown graphs of level 2. In Marc Bezem, editor, *CSL*, volume 12 of *LIPIcs*, pages 322–336. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

8 Alexander Kartzow and Paweł Parys. Strictness of the collapsible pushdown hierarchy. In preparation, 2012.

9 Teodor Knapik, Damian Niwinski, and Pawel Urzyczyn. Higher-order pushdown trees are easy. In Mogens Nielsen and Uffe Engberg, editors, *FoSSaCS*, volume 2303 of *Lecture Notes in Computer Science*, pages 205–222. Springer, 2002.

10 A. N. Maslov. The hierarchy of indexed languages of an arbitrary level. *Soviet Math. Dokl.*, 15:1170–1174, 1974.

11 Paweł Parys. The pumping lemma is incorrect? Unpublished, 2010.

Algorithmic Meta Theorems for Circuit Classes of Constant and Logarithmic Depth

Michael Elberfeld, Andreas Jakoby, and Till Tantau

Institut für Theoretische Informatik, Universität zu Lübeck
D-23538 Lübeck, Germany
{elberfeld,jakoby,tantau}@tcs.uni-luebeck.de

Abstract

An algorithmic meta theorem for a logic and a class C of structures states that all problems expressible in this logic can be solved efficiently for inputs from C. The prime example is Courcelle's Theorem, which states that monadic second-order (MSO) definable problems are linear-time solvable on graphs of bounded tree width. We contribute new algorithmic meta theorems, which state that MSO-definable problems are (a) solvable by uniform constant-depth circuit families (AC^0 for decision problems and TC^0 for counting problems) when restricted to input structures of bounded tree depth and (b) solvable by uniform logarithmic-depth circuit families (NC^1 for decision problems and $\#NC^1$ for counting problems) when a tree decomposition of bounded width in term representation is part of the input. Applications of our theorems include a TC^0-completeness proof for the unary version of integer linear programming with a fixed number of equations and extensions of a recent result that counting the number of accepting paths of a visible pushdown automaton lies in $\#NC^1$. Our main technical contributions are a new tree automata model for unordered, unranked, labeled trees; a method for representing the tree automata's computations algebraically using convolution circuits; and a lemma on computing balanced width-3 tree decompositions of trees in TC^0, which encapsulates most of the technical difficulties surrounding earlier results connecting tree automata and NC^1.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases algorithmic meta theorem, monadic second-order logic, circuit complexity, tree width, tree depth

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.66

1 Introduction

Courcelle's Theorem [6] states that every monadic second-order (MSO) definable problem can be decided in linear time on graphs of bounded tree width. Since many important graph properties are expressible in this logic, Courcelle's Theorem yields a unified framework for showing that numerous problems on graphs of bounded tree width are solvable in linear time. Recently we showed that both Courcelle's Theorem as well as its later extensions [1] also hold when "linear time" is replaced by "logarithmic space" [9], making the power of MSO-definability available for the study of logarithmic space.

The present paper furthers this line of research and transfers the idea of unified MSO-based problem definitions to circuit classes inside logarithmic space. During the course of this paper we identify MSO-based algorithmic meta theorems that place problems in the circuit classes AC^0, $GapAC^0$, TC^0, NC^1, and $\#NC^1$. The classes AC^0, $GapAC^0$, and TC^0 are defined via Boolean (AC^0), arithmetic ($GapAC^0$), and threshold (TC^0) circuit families of constant depth and unbounded fan-in. The classes NC^1 and $\#NC^1$ are defined via Boolean and arithmetic

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 66–77

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


circuits, respectively, of logarithmic depth and bounded fan-in. All our results concerning circuit classes hold for a strict form of uniformity, namely DLOGTIME-uniformity.

The inputs for Courcelle's Theorem are graphs of bounded tree width and many MSO-definable problems on such graphs are complete for logarithmic space, including even the question of whether the graph has a certain tree width [9], but also the reachability problem for trees. Thus, algorithmic meta theorems that place problems inside subclasses of L either need to restrict the logic or the kinds of inputs allowed. In the present paper, we consider the latter case: For the constant-depth circuit classes, we only allow input graphs that have *bounded tree depth* (a restriction of bounded tree width). For the logarithmic-depth circuit classes we allow arbitrary graphs of bounded tree width as input, but require that the graphs are *accompanied by tree decompositions in term representation*.

Bounded Tree Depth Structures and Constant-Depth Circuits Our first contribution is a set of meta theorems that place problems in constant-depth circuit classes. The inputs for these theorems are structures that have bounded tree *depth*, a measure on graphs that was introduced by Nešetřil and Ossona de Mendez [16] to quantify the similarity of graphs to star graphs (in opposition to tree width, which quantifies the similarity of graphs to trees). Characterizations of when a class C of graphs has bounded tree depth include: (a) All graphs in C have a tree decomposition of both bounded width and depth; or alternatively (b) all graphs in C have bounded longest path length. The tree depth of a logical structure is the tree depth of its Gaifman graph.

▶ **Theorem 1** (Decision Using Boolean Constant-Depth Circuits). *For every* MSO*-formula* ϕ *over some signature* τ *and every* $d \in \mathbb{N}$*, there is a* DLOGTIME*-uniform* AC^0*-circuit family that, on input of an arbitrary* τ*-structure* $\mathcal{S}$*, outputs* 1 *if, and only if, the tree depth of* $\mathcal{S}$ *is at most* d *and* $\mathcal{S} \models \phi$ *holds.*

As an example application, consider the problem of deciding whether a graph has a perfect matching. The complexity of this MSO-definable problem has been studied in detail and its complexity varies in dependence of the class of graphs under consideration. By the above theorem, deciding whether a graph of bounded tree depth has a perfect matching lies in AC^0. In contrast, it is known that the same problem for graphs of bounded tree *width* is L-complete [7, 9].

Instead of just deciding whether a formula is satisfied by a logical structure, when the formula has a free second-order variable, we can try to count the number of assignments of sets to the free variable that make the formula true. Moreover, if we count the number of solutions with respect to the sizes of these sets, this leads to *cardinality versions* of Courcelle's Theorem. These cardinality versions allow a much broader range of applications than the decision version and we will show how both known results from the literature and also new results can be proved in an elegant manner using these versions. To formulate the cardinality versions, we need a bit of terminology: Let $\phi(X_1, \dots, X_\ell, Y_1, \dots, Y_k)$ be an MSO-formula with two sets of free set variables, namely the X_i and the Y_j, and let $\mathcal{S}$ be a logical structure with universe S. The *solution histogram* of ϕ and $\mathcal{S}$, denoted by $\mathrm{hist}(\mathcal{S}, \phi)$, is an ℓ-dimensional integer array that tells us "how many solutions of a certain size exist". In detail, let $s = (s_1, \dots, s_\ell) \in \{0, \dots, |S|\}^\ell$ be an index vector that prescribes sizes for the sets that are substituted for the X_i. Then $\mathrm{hist}(\mathcal{S}, \phi)[s]$ equals the number of subsets $S_1, \dots, S_\ell, S'_1, \dots, S'_k \subseteq S$ with $|S_1| = s_1, \dots, |S_\ell| = s_\ell$ and $\mathcal{S} \models \phi(S_1, \dots, S_\ell, S'_1, \dots, S'_k)$. In other words, we count how often ϕ can be satisfied when the sets assigned to the X_i-variables have certain sizes, but impose no restrictions on the sizes of the Y_j. As a first example, let

$\phi_{\text{dom}}(X_1) = \forall x\big(X_1(x) \vee \exists y(X_1(y) \wedge E(y,x))\big)$, which expresses that X_1 is a dominating set in a graph with edge relation E. Then $\text{hist}(\mathcal{G}, \phi_{\text{dom}})[s_1]$ is the number of dominating sets of size s_1 in the graph $\mathcal{G}$. As a second example, let $\phi_{\text{matching}}(Y_1)$ be the formula expressing that Y_1 is an edge set that is a perfect matching in G. Then, since $\ell = 0$, the histogram $\text{hist}(\mathcal{G}, \phi_{\text{matching}})$ is just a scalar value that tells us how many perfect matchings G has.

In order to represent a histogram h using a single number $\text{num}(h) \in \mathbb{N}$, imagine h to be stored in computer memory with a word size large enough so that each of its entries fits into one memory cell (choosing the word size as $k|S|$ will suffice). Then $\text{num}(h)$ is the single number representing the whole of the memory contents (a formal definition of $\text{num}(h)$ will be given later). In particular, the bits of any single entry of h can easily be obtained from the bits of $\text{num}(h)$.

▶ **Theorem 2** (Histogram Computation Using Arithmetic Constant-Depth Circuits). *For every* MSO*-formula $\phi(X_1, \ldots, X_\ell, Y_1, \ldots, Y_k)$ over some signature τ and every $d \in \mathbb{N}$, there is a* DLOGTIME*-uniform* GapAC^0*-circuit family that, on input of a τ-structure $\mathcal{S}$ of tree depth at most d, outputs* $\text{num}(\text{hist}(\mathcal{S}, \phi))$.

By Theorem 1 we can check in AC^0 whether an input structure $\mathcal{S}$ has tree depth d and, if not, we could output an error value like -1. Applying Theorem 2 to the formula $\text{hist}(\mathcal{G}, \phi_{\text{matching}})$ shows that counting the number of perfect matchings in graphs of bounded tree depths lies in GapAC^0. Since GapAC^0 is contained in FTC^0, the functional version of the class TC^0 of constant-depth circuits with threshold gates, computing a particular bit of the number $\text{num}(\text{hist}(\mathcal{S}, \phi))$ can be done using a TC^0-circuit:

▶ **Corollary 3.** *For every* MSO*-formula $\phi(X_1, \ldots, X_\ell, Y_1, \ldots, Y_k)$ over some signature τ and every $d \in \mathbb{N}$, there is a* DLOGTIME*-uniform* TC^0*-circuit family that, on input of a τ-structure $\mathcal{S}$ of tree depth at most d, an ℓ-dimensional index s, and a bit position i, outputs the ith bit of* $\text{hist}(\mathcal{S}, \phi)[s]$.

We cannot hope to place the computation of solution histograms in any complexity class smaller than FTC^0 since the TC^0-complete problem MAJORITY is easily expressible using an MSO-formula and the histogram: Turning a string s into a logical structure $\mathcal{S} = (\{1, \ldots, |s|\}, P_1^{\mathcal{S}})$ in the usual manner by setting $i \in P_1^{\mathcal{S}} \Leftrightarrow s[i] = 1$, for the MSO-formula $\phi(X_1) = \forall x(X_1(x) \to P_1(x))$ more than half of the input bits are 1 if, and only if, $\text{hist}(\mathcal{S}, \phi)\big[\lfloor |s|/2 \rfloor + 1\big] > 0$. In Section 3 we use extensions of this idea to prove the TC^0-completeness of the unary version of integer linear programming with a constant number of equations.

Bounded Tree Width, Term Representations, and Logarithmic-Depth Circuits Our second contribution are algorithmic meta theorems for NC^1 and its arithmetic companion class $\#\text{NC}^1$. For these theorems the input structure is equipped with a tree decomposition of bounded width (no longer of bounded depth, though) given in term representation. The term representation of a tree like  is the string [[ ] [[ ] [ ]]], which exhibits the tree's ancestor relation.

▶ **Theorem 4** (Decision Using Boolean Logarithmic-Depth Circuits). *For every* MSO*-formula ϕ over some signature τ and every $w \in \mathbb{N}$, there is a* DLOGTIME*-uniform* NC^1*-circuit family that, on input of a τ-structure $\mathcal{S}$ along with a width-w tree decomposition in term representation for $\mathcal{S}$, decides whether $\mathcal{S} \models \phi$ holds.*

As an example application, consider the problem of deciding the language accepted by a tree automaton. It is well known that every such language lies in NC^1 [15]. The above

theorem allows us to reprove this fact succinctly: an MSO-formula can easily check (using existential second-order quantifiers) whether there is an assignment of states to the nodes of the tree that is locally consistent and that makes the automaton accept.

▶ **Theorem 5** (Histogram Computation Using Arithmetic Logarithmic-Depth Circuits). *For every* MSO*-formula* $\phi(X_1, \ldots, X_\ell, Y_1, \ldots, Y_k)$ *over some signature* τ *and every* $w \in \mathbb{N}$*, there is a* DLOGTIME*-uniform* $\#\mathrm{NC}^1$*-circuit family that, on input of a* τ*-structure* $\mathcal{S}$ *along with a width-*w *tree decomposition in term representation for* $\mathcal{S}$*, outputs* $\mathrm{num}(\mathrm{hist}(\mathcal{S}, \phi))$.

An application of this theorem is to count the number of accepting paths of nondeterministic visible pushdown automata.

Technical Contributions The proofs of the algorithmic meta theorems for constant-depth circuits rest on two new technical tools. First, we introduce a new model of automata, which we call *multiset automata*, that exactly captures the MSO-definable problems on unordered unranked labeled trees. Standard automata-theoretic approaches to proving meta theorems cannot be applied in the context of constant-depth circuits: all known approaches include preprocessing steps that enlarge the depth of the input trees by at least a logarithmic factor, making them infeasible for simulation by constant-depth circuits. Second, we develop algebraic representations of the computations of multiset automata using arithmetic circuits that keep track of the number of ways in which states can be reached.

In the context of research on logarithmic-depth circuits, trees in term representation are a natural form of input. In many papers (including the present), a central problem is that a logarithmic-depth circuit cannot work on the term representation directly when it has large depth. The standard approach is to recursively divide the tree into parts smaller by some constant factor, but doing so uniformly is an involved problem. We present a new algorithm for dealing with trees of arbitrary depth: It takes a tree T as input and outputs a width-3 tree decomposition of T that is perfectly balanced and, hence, has logarithmic depth. The bags of this decomposition form a hierarchical separation of T into subtrees along which a recursive algorithm can work. A key property of our construction is that it can be performed in TC^0.

Related Work Algorithmic meta theorems for monadic second-order logic have been studied intensively from the perspective of achieving a low runtime (see [12] for an overview), but there is less work on meta theorems that lead to exact classifications in complexity theoretic terms. Two exceptions are Wanke's paper [18], which shows that all problems that are captured by Courcelle's Theorem are in LOGCFL, and our paper [9], which places these problems further down into L.

Tree automata-based techniques are routinely used to prove time- and space-efficient variants of Courcelle's Theorem [1, 9]. The problem of deciding whether a fixed tree automaton accepts a given tree in term representation lies in NC^1 both in the ranked [15] and the unranked case [11].

Buss [3] used pebbling-based strategies to evaluate Boolean sentences in uniform NC^1. His method was later adopted to evaluate arithmetic sentences [2] and, more recently, to prove that the number of accepting computations of nondeterministic visible pushdown automata can be counted in $\#\mathrm{NC}^1$ [13].

Organization of This Paper After discussing the logical, graph theoretic and complexity theoretic background of our work in Section 2, in Sections 3 and 4 we sketch the proofs and

applications of the algorithmic meta theorems for constant-depth and logarithmic-depth circuits, respectively. Due to lack of space, formal proofs are omitted from the present conference paper; they can be found in its technical report version [10].

2 Background

A detailed review of the notations on logic, graphs, tree decompositions, and complexity classes that we use in the present paper can be found in our technical report [10]. In the following, we just point out less common notations that we will use: For a graph G, we write $V(G)$ for its vertex set and $E(G) \subseteq V(G) \times V(G)$ for its edge set. We consider trees as special cases of directed graphs, where edges point from the root towards the leaves, but call their vertices *nodes*. A tree decomposition is a pair (T_D, B_D) where T_D is a tree and B_D is a labeling function $B_D \colon V(T_D) \to \mathcal{P}(V(G))$, where $\mathcal{P}(X)$ is the power set of X, that satisfies standard connectedness and edge covering conditions. The *closure* $\mathrm{clos}(T)$ of a directed tree T is the graph with vertex set $V(\mathrm{clos}(T)) = V(T)$ and edge set $E(\mathrm{clos}(T)) = \{(v,w) \in V(T) \times V(T) \mid \text{there is a } v\text{-to-}w \text{ or a } w\text{-to-}v \text{ path in } T\}$. The *tree depth* [16] of a connected graph G, denoted by $\mathrm{td}(G)$, is 1 plus the minimum depth of a rooted tree T with $V(G) = V(T)$ and $E(G) \subseteq E(\mathrm{clos}(T))$. The tree depth of a graph with components $C_1, \ldots, C_m$ is $\max_{i \in \{1,\ldots,m\}} \mathrm{td}(C_i)$. The tree width and tree depth of a logical structure are those of its Gaifman graph. The *longest path length* $\mathrm{lpl}(G)$ of a graph is the length of the longest path in the graph. Nešetřil and Ossona de Mendez [17] showed that $\mathrm{lpl}(G) \leq 2^{\mathrm{td}(G)} - 2$ holds for each undirected graph G.

3 Algorithmic Meta Theorems For Constant-Depth Circuit Classes

In the present section we prove the algorithmic meta theorems that relate monadic second-order properties of graphs of bounded tree depth to constant-depth circuit classes (Theorems 1 and 2 from the introduction). The route toward proving them is the following: (1) First, we show how a tree decomposition of a logical structure of bounded tree depth can be computed using first-order reductions. Once available, we show how to adjust the original MSO-formula to an equivalent formula for the computed tree. This first step allows us to replace the task of computing solution histograms for structures of any signature by the more manageable problem of computing solution histograms for trees. (2) Second, we introduce the notion of multiset automata for unordered unranked labeled trees, prove standard closure properties for these automata, and show that they capture exactly the MSO-definable properties of unordered unranked labeled trees. This turns the problem of deciding formulas into the problem of evaluating multiset automata. (3) After that we explain how to reduce computing the number of ways in which multiset tree automata accept an input tree to evaluating arithmetic circuits of constant depth. In the course of this step, we address the problem of how histograms can be encoded as numbers. As we will see, by using an appropriate encoding, we may assume that our formulas ϕ are all of the form $\phi(X_1, \ldots, X_k)$, that is, we may assume that no variables Y_i are present. This is why the lemmas and theorems of the present section are all formulated without references to any Y_j. (4) At the end, we apply the algorithmic meta theorems to concrete problems. We show, in particular, that the unary version of integer linear programming with a constant number of equations is complete for TC^0.

Turning Tree-Depth-Bounded Structures into Depth-Bounded Tree Structures The first step toward our goal of proving Theorems 1 and 2 is to compute tree decompositions of bounded width and depth for input structures of bounded tree depth using first-order reductions.

▶ **Lemma 6.** *Let τ be a signature and $d \in \mathbb{N}$. There is a first-order computable function that, on input of the encoding* $\mathrm{code}(\mathcal{S})$ *of a τ-structure $\mathcal{S}$, outputs either (a) a tree decomposition D of $\mathcal{S}$ of width at most $2^d - 3$ and depth at most $2^d - 1$, or (b) "no" and* $\mathrm{td}(\mathcal{S}) > d$ *holds in this case.*

The following lemma uses a first-order reduction to transform the task of computing histograms for input structure of any signature to the task of computing histograms for tree structures. For the formulation of the lemma, we use the following terminology: An *s-tree structure* is a structure $\mathcal{T} = (V, E^{\mathcal{T}}, P_1^{\mathcal{T}}, \ldots, P_s^{\mathcal{T}})$ over the signature $\tau_{s\text{-tree}} = \{E^2, P_1^1, \ldots, P_s^1\}$ where $(V, E^{\mathcal{T}})$ is a directed tree.

▶ **Lemma 7.** *Let $\phi(X_1, \ldots, X_\ell)$ be an* MSO*-formula over some signature τ and $w \in \mathbb{N}$. There is an $s \in \mathbb{N}$, a* MSO*-formula $\psi(X_1, \ldots, X_\ell)$ over $\tau_{s\text{-tree}}$, and a first-order computable function that, on input of any τ-structure $\mathcal{S}$ with universe S and a width-w tree decomposition $D = (T_D, B_D)$ for $\mathcal{S}$, produces an s-tree structure $\mathcal{T}$, such that (a) the depth of $\mathcal{T}$ equals the depth of T_D plus 1, and (b) for all indices $i \in \{0, \ldots, |S|\}^\ell$ we have* $\mathrm{hist}(\mathcal{S}, \phi)[i] = \mathrm{hist}(\mathcal{T}, \psi)[i]$ *and all other entries in the array* $\mathrm{hist}(\mathcal{T}, \psi)$ *are 0.*

Lemma 6 and Lemma 7 together provide a transformation from evaluating MSO-formulas on logical structures of bounded tree depth to evaluating them on s-tree structures of bounded depth.

Turning Formulas on Tree Structures into Tree Automata The trees underlying the s-tree structures that are produced by Lemma 7 do not impose an order on sibling nodes and nodes may have an unbounded number of children. Such trees, with the s unary predicates represented by binary strings, are known as *unordered, unranked* labeled trees in the literature [14]. "Unordered" means that there is no total order on sibling nodes and "unranked" stands for unbounded degree. In this section we introduce a new notion of automata that is appropriate for unordered labeled trees and prove that it exactly captures the MSO-definable properties of unordered labeled trees, resulting in a theorem which can be seen as an extension of the classical Büchi–Elgot–Trakhtenbrot Theorem. Moreover, the translation between MSO-formula and automata will preserve the sizes and number of solutions, thereby establishing a reduction from computing solution histograms for MSO-formulas on s-tree structures to evaluating tree automata.

Tree-automata-based proofs of time and space efficient variants of Courcelle's Theorem transform input structures into trees where the underlying tree has bounded degree. Then, in these proofs MSO-formulas on bounded degree trees are transformed into the classical tree automata for ranked labeled trees that were developed in the 1970's. Adopting this strategy and transforming s-tree structures with unbounded degree into tree structures of bounded degree would come at the cost of increasing the depth of the tree by at least a logarithmic factor and this would imply vertical data dependencies in the tree that we cannot hope to handle with constant-depth circuits. Due to this reason, we need an automaton model that does not force us to change the topology of the tree. For a similar reason, we cannot use some order on the children and translate to the tree automata for ordered unranked trees that are studied in the context of XML processing [11]; here the horizontal data dependencies on

sibling subtrees are too high. In fact, such automata are able to decide any regular property on the ordered children of a node and, thus, cannot be simulated by constant-depth circuits.

The only automaton model from the literature that does not introduce dependencies between nodes that cannot be handled by constant-depth circuits is due to Libkin [14] who defined *counting unranked tree automata*, which are equivalent to MSO on unordered trees. The transition functions of these automata are defined in terms of Boolean functions: they allow us to assign a state q' to a node with symbol σ if a Boolean function $\delta(\sigma, q')$, which depends on the number of occurrences of states at the children, evaluates to 1. However, it is unclear (at least to us) how these automata could be used to compute solution histograms since we need to relate the states assigned to the subtrees of a node with the state that is assigned to the whole tree in a transparent way, without "hiding it inside a Boolean function."

In this section, we develop multiset automata as a notion that exactly captures the MSO-definable properties of unordered labeled trees (unranked or ranked) and that allows us to control the assignment of states to the children of a node such that we can later establish a cardinality-preserving transformation into arithmetic circuits.

A *multiset* M on a universe U is a function $\#_M \colon U \to \mathbb{N}$ that assigns a *multiplicity* to each element of U. We write $\mathcal{P}_\omega(U)$ for the class of all multisets on U and we write $\mathcal{P}_m(U)$ for the class of all multisets on U where each element has multiplicity at most m. Given a number $m \in \mathbb{N}$, let $M|_m$ be $\#_{M|_m}(e) = \min\{\#_M(e), m\}$ for $e \in U$. We call $M|_m$ the *capped version of M to multiplicity m*.

▶ **Definition 8** (Multiset Automata)**.** A *nondeterministic (bottom-up) multiset automaton* is a tuple $A = (\Sigma, Q, Q_a, \Delta)$ consisting of an *alphabet* Σ, a *state set* Q, a set $Q_a \subseteq Q$ of *accepting states*, and a *state transition relation* $\Delta \subseteq \Sigma \times \mathcal{P}_m(Q) \times Q$ for some constant *multiplicity bound* m. The automaton is *deterministic* if for every $\sigma \in \Sigma$ and every $M \in \mathcal{P}_m(Q)$ there is exactly one $q \in Q$ with $(\sigma, M, q) \in \Delta$; in this case we can view Δ as a *state transition function* $\delta \colon \Sigma \times \mathcal{P}_m(Q) \to Q$.

▶ **Definition 9** (Computation of a Multiset Automaton)**.** Let (T, l) be a labeled tree, where $l \colon V(T) \to \Sigma$ is the labelling function, and let $A = (\Sigma, Q, Q_a, \Delta)$ be a multiset automaton. A *computation* of A on (T, l) is a partial assignment $q \colon V(T) \to Q$ such that for every node $n \in V(T)$ for which $q(n)$ is defined, we have that (a) the value $q(c)$ is defined for each child c of n in T and (b) for the multiset $M = \{q(c) \mid c \text{ is a child of } n\}$ we have $(l(n), M|_m, q(n)) \in \Delta$. A computation is *accepting*, if $q(r) \in Q_a$ holds for the root node r of T. The *tree language* $L(A)$ contains all labeled trees accepted by A.

Given an s-tree structure $\mathcal{T} = (V, E^{\mathcal{T}}, P_1^{\mathcal{T}}, \ldots, P_s^{\mathcal{T}})$ and sets $S_1, \ldots, S_\ell \subseteq V$, let us write $T(\mathcal{T}, S_1, \ldots, S_\ell)$ for the labeled tree whose node set is V, whose edge set is $E^{\mathcal{T}}$, and whose labeling function maps each node $v \in V$ to the bitstring $l_1 \ldots l_s x_1 \ldots x_\ell \in \{0,1\}^{s+\ell}$ with $l_i = 1 \Leftrightarrow v \in P_i^{\mathcal{T}}$ and $x_i = 1 \Leftrightarrow v \in S_i$. We write $T(\mathcal{T})$ in case $\ell = 0$.

▶ **Theorem 10.** *Let $s, \ell \in \mathbb{N}$.*

1. *For every* MSO*-formula $\phi(X_1, \ldots, X_\ell)$ over $\tau_{s\text{-tree}}$ there is a multiset automaton A with alphabet $\{0,1\}^{s+\ell}$, such that for all s-tree structures $\mathcal{T}$ with universe V and $S_1, \ldots, S_\ell \subseteq V$ we have $\mathcal{T} \models \phi(S_1, \ldots, S_\ell)$ if, and only if, A accepts $T(\mathcal{T}, S_1, \ldots, S_\ell)$.*
2. *For every multiset automaton A with alphabet $\{0,1\}^{s+\ell}$ there is an* MSO*-formula $\phi(X_1, \ldots, X_\ell)$ over $\tau_{s\text{-tree}}$, such that for all s-tree structures $\mathcal{T}$ with universe V and $S_1, \ldots, S_\ell \subseteq V$ we have $\mathcal{T} \models \phi(S_1, \ldots, S_\ell)$ if, and only if, A accepts $T(\mathcal{T}, S_1, \ldots, S_\ell)$.*

Our proof of the theorem follows Arnborg et al. [1], but modified to unranked trees rather than ranked trees and multiset automata rather than usual tree automata. It entails proofs

of standard closure properties: The class of tree languages accepted by multiset automata is closed under intersection, union, complement, and for every nondeterministic multiset automaton there is a deterministic automaton accepting the same tree language.

From Automaton Evaluation to Arithmetic Circuit Evaluation Theorem 10 shows that in order to decide whether a given MSO-formula is true for a given tree, we can instead evaluate a multiset automaton. Since any logical structure of bounded tree depth can be transformed into a labeled tree of constant depth, we have all the ingredients together to prove Theorem 1 from the introduction.

Instead of just *deciding* formulas, in the remaining part of this section we turn our attention to the more challenging problem of computing the solution histograms. Our aim will be to replace the evaluation of automata by the evaluation of convolution circuits, see Lemma 11, such that the circuits's outputs are the sought solution histograms. Then we reduce the evaluation of convolution circuits to the evaluation of arithmetic circuits. Since arithmetic circuits of constant depth can be evaluated in GapAC^0, we get Theorem 2.

Theorem 10 establishes a link between formulas and multiset automata that is "solution-preserving" in the sense that there is a one-to-one correspondence between satisfying assignments to the free variables of the formulas and labelings of the trees that make an automaton A accept. In order to talk more easily about the number of such labelings, we recall the notion of *multicolorings* from [9]: Given a set S, a *multicoloring* of S is a tuple $(S_1, \dots, S_\ell)$ of subsets $S_j \subseteq S$ for $j \in \{1, \dots, \ell\}$. Given a set X of multicolorings of S, let $\mathrm{hist}(X)$ denote the $[|S|+1]^\ell$-array whose entry at index $i = (i_1, \dots, i_\ell)$ is the number of multicolorings $(S_1, \dots, S_\ell) \in X$ with $|S_1| = i_1$, …, $|S_\ell| = i_\ell$. Given a multiset automaton $A = (\{0,1\}^{s+\ell}, Q, Q_a, \Delta)$ and an s-tree structure $\mathcal{T}$ with universe V, let us write $S_A(\mathcal{T}, P)$ for the set of tuples $(S_1, \dots, S_\ell)$ with $S_i \subseteq V$ for which A reaches a state $q \in P$ at the root of $T(\mathcal{T}, S_1, \dots, S_\ell)$. Clearly, $S_A(\mathcal{T}, P)$ is a set of multicolorings of V. In particular, for the automaton A constructed in Theorem 10 for a formula ϕ we have $\mathrm{hist}(\mathcal{T}, \phi) = \mathrm{hist}(S_A(\mathcal{T}, Q_a))$. This means that "all" we have to do is to devise a way of computing $\mathrm{hist}(S_A(\mathcal{T}, Q_a))$ for a given automaton A and a tree $\mathcal{T}$.

For the computation of $\mathrm{hist}(S_A(\mathcal{T}, Q_a))$ we use *convolution circuits*, which are similar to arithmetic circuits, only instead of numbers whole histogram arrays are passed between gates. The basic gates of a convolution circuit are addition gates (which just add the arrays componentwise), subtraction gates (if there are no subtraction gates, the circuit is called *positive*), and convolution gates. The convolution $C = A * B$ of two arrays A and B is defined by $C[k] = \sum_{i \in [r]^\ell, j \in [s]^\ell \text{ with } k = i+j} A[i]B[j]$. The addition of two histograms corresponds exactly to combining two disjoint sets of solutions for the same tree, while the convolution of the histograms corresponds to combining the solutions of two sibling subtrees.

The construction of convolution circuits for a ranked automata is already described in [9] for the logspace setting. For the unranked automata considered in the present section, the construction needs to be more involved: For a node of a tree with a large number of children, the difficult part is to combine the histograms of all of these children so that they correspond to some particular capped version of the multiset of states reached at the children. The details of the recursive construction that achieves this can be found in our technical report [10]. The main result established is the following, where $\mathrm{val}(C)$ is the output of C:

▶ **Lemma 11.** *Let $A = (\{0,1\}^{s+\ell}, Q, Q_a, \delta)$ be a deterministic multiset automaton with multiplicity bound $m \in \mathbb{N}$. Then there is a first-order computable function that maps every s-tree structure $\mathcal{T} = (V, P_1^{\mathcal{T}}, \dots, P_s^{\mathcal{T}})$ to a convolution circuit C such that (a) $\mathrm{val}(C) = \mathrm{hist}(S_A(\mathcal{T}, Q_a))$, (b) the depth of C is bounded by a function that depends on A and linearly*

on the depth of $\mathcal{T}$, and (c) the fan-in of C is bounded by a function that depends on A and linearly on the degree of $\mathcal{T}$. Furthermore, if the degree of $\mathcal{T}$ is bounded by m, then C is positive.

The final problem is to move from convolution circuits to arithmetic circuits. This is quite easy to achieve: For a vector $b = (b_1, \ldots, b_\ell)$ of large bases and an ℓ-dimensional histogram h, set $\mathrm{num}_b(h) = \sum_{(i_1,\ldots,i_\ell)\in\{0,\ldots,|S|\}^\ell} h[i_1, \ldots, i_\ell] b_1^{i_1} \cdots b_\ell^{i_\ell}$. Then $\mathrm{num}_b(A * B) = \mathrm{num}_b(A) \cdot \mathrm{num}_b(B)$. Thus, if we replace all constants c in the circuit C from the above lemma by $\mathrm{num}_b(c)$ and replace all convolution gates by multiplication gates, we get the arithmetic circuit claimed in Theorem 2.

Application: Placing Problems in Constant-Depth Circuit Classes The algorithmic meta theorems developed in this section allow putting problems into the uniform circuit classes AC^0, GapAC^0 and TC^0 by using direct MSO-based definitions of problems on structures of bounded tree depth or reductions to MSO-definable problems on bounded tree depth structures.

▶ **Theorem 12.** *For every $d \in \mathbb{N}$, the language $\{(G, s) \mid G$ has tree depth at most d and at least s perfect matchings$\}$ is TC^0-complete under AC^0-reductions.*

▶ **Theorem 13.** *For each $\ell \in \mathbb{N}$, the problem ℓ-INTEGER-LINEAR-PROGRAMMING, the version of integer linear programming where there are at most ℓ equations and the input numbers are given in unary, is complete for TC^0 under AC^0-reductions.*

4 Algorithmic Meta Theorems For Logarithmic-Depth Circuit Classes

In the present section we prove Theorems 4 and 5, which involve circuits of *logarithmic* depth rather than constant depth as in the previous section. The inputs now consist of (an encoding of) a logical structures $\mathcal{S}$ together with a tree decomposition D of $\mathcal{S}$, where T_D is given in term representation. The proofs follow along the same lines as those of Theorems 1 and 2, which involved the following steps: (1) Compute a tree decomposition of the input structure and move from formulas on the input structures to formulas on trees, (2) move from formulas on trees to the evaluation of tree automata, (3) move from the evaluation of tree automata to convolution circuits and from convolution circuits to arithmetic circuits. Clearly, computing a tree decomposition is no longer necessary since it is already part of the input. All of the other steps are also possible when the tree depth is no longer constant, the resulting circuits then simply have arbitrary depth. Since it is known that tree automata can be evaluated in NC^1 on trees given in term representation [15, 11], Theorem 4 follows (almost) immediately from our previous arguments.

The main obstacle in proving Theorem 5 is that one can evaluate arithmetic *formulas* of arbitrary depth in $\#\mathrm{NC}^1$ [2, 5], but evaluating arithmetic *circuits* can be done in $\#\mathrm{NC}^1$ only if the circuit has logarithmic depth (evaluating arithmetic circuits of arbitrary depth is already FP-hard when we cap the numbers to enforce the outputs to have only polynomial length, which they need not have in general). This means that, at some point in the course of the proof of Theorem 5, we need to move from trees or circuits of arbitrary depth to logarithmic depth. Previous papers, such as [13], have faced a similar obstacle, namely evaluating tree-like structures of arbitrary depth whose nodes perform a complicated algebraic operation on the values of their children. In these papers, the approach was to somehow extend the ideas used in the proof that evaluating arithmetic formulas can be done in $\#\mathrm{NC}^1$ [2, 5] to more general algebraic structures.

Our approach to tackling this problem is different and may be of independent interest. Rather than trying to adapt algorithms to the convolution computations that would be needed in our setting, we attack the problem at a much earlier stage: We balance the tree decomposition. Since all of our later algorithms do not increase the depth of the considered trees, we get the desired arithmetic circuits of logarithmic depth. In detail, we show how a balanced width-3 tree decomposition of an arbitrary tree can be computed using constant-depth threshold circuits. The construction has two key properties. First, it is based on the classical tree contraction method, which is used a lot in the context of parallel random access machines, but which hitherto was not used in the context of NC^1. Using it will allow us to compute a balanced tree decomposition even in TC^0 and not only in NC^1. Second, the tree decomposition we compute does *not* have the property that the nodes of each bag form a balanced separator of some part of the tree. Normally, recursive NC^1 algorithms find sets of nodes that in each step split up the tree into components that are smaller than the current tree by a certain factor. This is not the case for the sets of nodes in our bags: While we *can* ensure that the whole tree is balanced and, hence, has logarithmic depth, we *cannot* ensure that the elements of any individual bag split the tree in some balanced way. Naturally, a lot of bags will have this balancing property (otherwise no tree decomposition of logarithmic depth would result), but we cannot say anything about where these balancing bags will lie. It seems that this more global approach (just find a tree decomposition of logarithmic depth) instead of the traditional local approach (find a balancing separator for each subtree recursively) allows us to lower the circuit complexity to a constant depth.

In the following, we first review term representations and, then, sketch the proof of Theorem 4. After that, we describe the technical result on how a width-3 tree decomposition of any tree can be computed in FTC^0; and then use this result to prove Theorem 5. At the end of this section, we sketch applications of the established meta theorems.

Background on Term and Ancestor Representations of Trees Up to now, the details of how tree decompositions are encoded as strings was not important; indeed, in the context of constant tree *depth* almost any encoding of the input graph and of tree decompositions will do since they can easily be transformed into one another. In the context of logarithmic-depth circuits, however, it is well known that it is crucial that the "ancestor relation" of the tree (for directed trees, this is exactly the transitive closure) is made accessible to the circuits, rather than just a pointer-structure or an adjacency matrix. There are two different ways of encoding this relation: Explicitly as a list of pairs or implicitly as a bracket structure. The two representations can be transformed into one another using TC^0-circuits and we will use both of them.

Decision by Logarithmic-Depth Circuits for Term Representations As mentioned earlier, the proof machinery established in Section 3 allows us already to prove Theorem 4 from the introduction. The only obstacle is that in all intermediate steps we do not only need to compute trees, but also their term representations. This is straightforward to achieve, see our technical report for details [10].

Computing Width-3 Tree Decompositions of Trees in Constant Depth We show that using only TC^0-circuits, for every tree T given in term representation we can compute a width-3 tree decomposition (T_D, B_D) of T (regarded as a graph) such that T_D is a perfectly balanced binary tree (and, hence, has logarithmic depth):

▶ **Theorem 14.** *There is a* DLOGTIME*-uniform* FTC^0*-circuit family that on input of a term representation of a tree T outputs a term representation of a width-3 tree decomposition (S, B) of T where S is a balanced binary tree.*

The proof idea is surprisingly simple: As was already implicitly observed by Buss [4], the trees resulting from the different stages of the classical tree contraction method can be computed in TC^0. During a tree contraction step, for a leaf n, one considers its sibling s, its parent p, and its grandparent g. We call $c = (n, s, p, g)$ a *contraction tuple* and associate a set $I(c)$ of nodes with it that covers all nodes that have been "contracted away" inside this tuple. Our two key observations are the following: (a) For every two contraction tuples c and c', the sets $I(c)$ and $I(c')$ are either disjoint or one is contained in the other. From this we can derive that the contraction tuples can be arranged in a tree of logarithmic depth. (b) If we attach the bag $\{n, s, p, g\}$ to each node (n, s, p, g) of this "tree of contraction tuples," we get a width-3 tree decomposition of the original tree.

Computing Histograms by Logarithmic-Depth Circuits for Term Representations Recall that our goal for the present section is to prove Theorem 5, that our line of proof was to do the same sequence of transformations as we did in Section 3 for constant depth circuits, and that the missing building block was a procedure to turn an arbitrary tree decomposition into a tree decomposition of logarithmic depth. Theorem 14 provides us with the tools to build this missing block.

Application: Placing Problems in Logarithmic-Depth Circuit Classes We discuss some examples of how to use the algorithmic meta theorems for logarithmic-depth circuit classes to put decision and counting problems into NC^1 and $\#\mathrm{NC}^1$, respectively. A simple example is the problem of evaluating Boolean sentences that are given as terms, a problem well known to lie in DLOGTIME-uniform NC^1 [3, 2].

Buss [3] extended his NC^1-approach for the evaluation of Boolean sentences to also cover the membership problem for parenthesis languages. Later researchers adapted this approach to show that larger classes of context-free languages can be decided in NC^1, with the most general one being the result of Dymond [8] that languages recognizable by *visible pushdown automata* are in NC^1. Besides deciding whether a string is accepted by a fixed VPA, recently the problem of counting the number of accepting computation paths of nondeterministic VPAs was studied in the context of logarithmic-depth circuits and shown to be complete for $\#\mathrm{NC}^1$ by Krebs, Limaye, and Mahajan [13]. Theorems 4 and 5 can be used to reprove that these decision and counting problems are in NC^1 and $\#\mathrm{NC}^1$, respectively [10].

5 Conclusion

In the present paper we transferred the idea of unifying the study of computational problems by using MSO-based problem definitions and tree decompositions to circuit complexity classes inside logarithmic space, leading to algorithmic meta theorems for Boolean and arithmetic circuit classes of constant and logarithmic depth. Regarding constant-depth circuits, we discussed how to put the problem of solving a linear equation system that contains a constant number of equations whose coefficient are given in unary into TC^0. The most general application for logarithmic-depth circuits showed an alternative proof of a recent result that one can count the number of accepting paths of visible pushdown automata in $\#\mathrm{NC}^1$.

A natural direction of further research would be to try and use our theorems for logarithmic-depth circuits to simulate some generalization of visible pushdown automata where the height of the stack at different positions in time can be computed in advance; say, in NC^1 instead of FTC^0. Another direction would be to find algorithmic meta theorems that unify problems lying in other complexity classes around logarithmic space. Such research would need to

address all three dimensions of algorithmic meta theorems: (a) the considered logic, (b) the considered class of input structures, and (c) the considered complexity class. We may go from MSO to more expressive or less expressive logics (like, for example, MSO on graphs where we can only quantify over vertex sets). Or we may consider other classes of structures that are more or less restrictive than bounded tree width (like, for example, bounded clique width).

References

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

2 S. Buss, S. Cook, A. Gupta, and V. Ramachandran. An optimal parallel algorithm for formula evaluation. *SIAM J. Comput.*, 21(4):755–780, 1992.

3 Samuel R. Buss. The boolean formula value problem is in ALOGTIME. In *Proceedings of STOC 1987*, pages 123–131. ACM, 1987.

4 Samuel R. Buss. Algorithms for boolean formula evaluation and for tree contraction. In Peter Clote and Jan Krajíček, editors, *Arithmetic, Proof Theory, and Computational Complexity*, pages 95–115. Oxford University Press, 1993.

5 Hervé Caussinus, Pierre McKenzie, Denis Thérien, and Heribert Vollmer. Nondeterministic NC^1 computation. *J. Comput. Syst. Sci.*, 57(2):200–212, 1998.

6 Brouno Courcelle. Graph rewriting: An algebraic and logic approach. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 193–242. Elsevier and MIT Press, 1990.

7 Bireswar Das, Samir Datta, and Prajakta Nimbhorkar. Log-space algorithms for paths and matchings in k-trees. In *Proceedings of STACS 2010*, volume 5 of *LIPIcs*, pages 215–226. Schloss Dagstuhl LZI, 2010.

8 Patrick Dymond. Input-driven languages are in $\log n$ depth. *Information Processing Letters*, 26(5):247–250, 1988.

9 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Logspace versions of the theorems of Bodlaender and Courcelle. In *Proceedings of FOCS 2010*, pages 143–152, 2010.

10 Michael Elberfeld, Andreas Jakoby, and Till Tantau. Algorithmic meta theorems for circuit classes of constant and logarithmic depth. Technical Report ECCC-TR11-128, 2011.

11 Georg Gottlob, Christoph Koch, Reinhard Pichler, and Luc Segoufin. The complexity of XPath query evaluation and XML typing. *J. ACM*, 52(2):284–335, 2005.

12 Martin Grohe and Stephan Kreutzer. Methods for algorithmic meta theorems. In *Model Theoretic Methods in Finite Combinatorics*, volume 558 of *Contemporary Mathematics*, pages 181–206. American Mathematical Society, 2011.

13 Andreas Krebs, Nutan Limaye, and Meena Mahajan. Counting paths in VPA is complete for $\#NC^1$. In *Proceedings of COCOON 2010*, volume 6196 of *LNCS*, pages 44–53. Springer, 2010.

14 Leonid Libkin. Logics for unranked trees: An overview. *Logical Methods in Computer Science*, 2(3), 2006.

15 Markus Lohrey. On the parallel complexity of tree automata. In *Proceedings of RTA 2001*, volume 2051 of *LNCS*, pages 201–215. Springer, 2001.

16 Jaroslav Nešetřil and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Combin.*, 27(6):1022—1041, 2006.

17 Jaroslav Nešetřil and Patrice Ossona de Mendez. Grad and classes with bounded expansion I. Decompositions. *Eur. J. Combin.*, 29(3):760–776, 2008.

18 Egon Wanke. Bounded tree-width and LOGCFL. *J. Algorithm*, 16(3):470–491, 1994.

An Approximation Algorithm for #k-SAT

Marc Thurley

Centre de Recerca Matemàtica, Bellaterra, Spain. **Supported by Marie Curie Intra-European Fellowship 271959**

Abstract

We present a simple randomized algorithm that approximates the number of satisfying assignments of Boolean formulas in conjunctive normal form. To the best of our knowledge this is the first algorithm which approximates #k-SAT for any $k \geq 3$ within a running time that is not only non-trivial, but also significantly better than that of the currently fastest exact algorithms for the problem. More precisely, our algorithm is a randomized approximation scheme whose running time depends polynomially on the error tolerance and is mildly exponential in the number n of variables of the input formula. For example, even stipulating sub-exponentially small error tolerance, the number of solutions to 3-CNF input formulas can be approximated in time $O(1.5366^n)$. For 4-CNF input the bound increases to $O(1.6155^n)$.

We further show how to obtain upper and lower bounds on the number of solutions to a CNF formula in a controllable way. Relaxing the requirements on the quality of the approximation, on k-CNF input we obtain significantly reduced running times in comparison to the above bounds.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics

Keywords and phrases #k-SAT, approximate counting, exponential algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.78

1 Introduction

The design and analysis of algorithms that determine the satisfiability or count the models of k-CNF formulas has quite some tradition. In the case of the satisfiability problem the earliest algorithm with a worst case running time which is significantly better than the trivial $\text{poly}(n)2^n$ bound dates back to at least 1985 [14]. The time bounds have improved gradually over the years with most recent results (only a few of which are [11, 18, 10, 9]) being analyses of randomized algorithms that have been obtained from either Schöning's algorithm [20], the algorithm of Paturi, Pudlák, Saks, and Zane [17], or a combination of both. The currently fastest algorithm for 3-SAT by Hertli [9] running in time $O(1.30704^n)$ falls roughly into the second category. The corresponding counting problems have seen similar improvements [3, 26, 2, 13, 4, 24] over the trivial time bound, with the current best worst case running time for #3-SAT being $O(1.6423^n)$ obtained by Kutzkov [13].

Quite surprisingly, however, the situation is completely different for the *approximation* of #k-SAT. To the best of my knowledge not even small improvements over the trivial worst case time bound are known.[1] This, however, does not seem to be due to a general lack of interest in the problem itself. From a complexity theoretical point of view, for example, several already classic papers [22, 23, 19] study questions closely related to direct algorithmic problems in #k-SAT approximation. In particular Valiant and Vazirani [23] bound the

[1] Disregarding, of course the pathological fact that exact methods can be interpreted as approximation algorithms, as well.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 78–87

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

complexity of the approximation problem from above by reduction to SAT, and hence settle its complexity in a certain sense.

While theoretical results on the approximation of #k-SAT are rather old, there are several heuristic approaches to the problem, that have all appeared only fairly recently. Motivated by questions of practicability, these results focus on methods that can be shown empirically to work well, while sacrificing some (at least theoretically) desirable properties. That is, some of these approaches yield approximations without any guarantee on the quality of the approximation [25, 5]. Others yield reliable lower and upper bounds [8, 7, 12] which, in certain cases, are surprisingly good although generally their quality is rather low. In particular, this line of work does not provide rigorous bounds on running times and neither does it yield rigorous quality estimates of the approximation computed.

With regard to the above results, the lack of competitive worst case bounds for #k-SAT approximation algorithms seems to be due to several factors. First of all the exact algorithms found in the literature and their analyses do not seem to carry over easily to the approximation problem. Secondly, complexity theoretical insights are usually not considered applicable in the context of designing fast exponential algorithms. An example is the technique of Valiant and Vazirani which leads to a significant blow up in formula size. And thirdly, it is not clear which of the known algorithmic ideas used in the heuristic approaches could at least in principle show a good worst case behavior.

1.1 Contributions

In this paper we will see that one can indeed not only improve upon the trivial worst-case time bound mentioned above. But the algorithm we will present also provides arbitrarily good precision in significantly less time than known exact methods. To be more precise, the algorithm we present is a *randomized approximation scheme* for #k-SAT for every $k \geq 3$. Given a freely adjustable error tolerance $\epsilon > 0$, randomized approximation schemes produce an output that is within a factor of e^{ϵ} of the number $\#F$ of solutions of some input formula F.

We obtain the following main result, which we state here only for $k = 3$ and $k = 4$. The general result will be discussed in Section 3.

▶ Theorem 1.1. There is a randomized approximation scheme running in time $O(\epsilon^{-2} \cdot 1.5366^n)$ for #3-SAT and in time $O(\epsilon^{-2} \cdot 1.6155^n)$ for #4-SAT.

For #3-SAT this algorithm is already significantly faster than the currently fastest exact algorithm from [13] which runs in time $O(1.6423^n)$. For #4-SAT the benefit of approximation is even more impressive, as the best bound for exact methods is still the $O(1.9275^n)$ bound of the basically identical algorithms of Dubois [3] and Zhang [26].

We will see that the algorithm of Theorem 1.1 is not complicated and monolithic like the branching algorithms usually employed in exact counting results. But it is actually a combination of two very simple and very different algorithms. The main reason for considering this combination relies on two pieces of intuition. On the one hand, if a formula has few solutions, then it is not too bad an idea to compute their number by simply enumerating them. On the other hand, if a formula has many solutions, then a quite trivial sampling algorithm should yield good results.

Observe that the result of Theorem 1.1 can already be used to compute e.g. $\#F$ exactly in time $O(1.5366^n)$ for any 3-CNF formula which has only a sub-exponential number of solutions. To achieve this we only have to set ϵ appropriately. However, we shall see below, that this can also be achieved in significantly less time. Motivated by the heuristic results on the

approximation of #k-SAT described above, we also study the effect of weaker requirements on the approximation bounds. It seems, of course, perfectly reasonable to assume that weaker bounds should come at the benefit of dramatically improved running time bounds. We will therefore show that this is the case. With respect to lower bounds we obtain:

▶ Theorem 1.2 (Lower Bound Algorithm). There is a randomized algorithm which, on input a 3-CNF formula F on n variables and a natural number N, performs the following in time $O(N^{0.614} \cdot 1.30704^n)$:

- If $\#F > N$ it reports this with probability at least 3/4.
- If $\#F \leq N$ then with probability at least 3/4 it reports this and outputs the correct value $\#F$.

Furthermore, there is a deterministic algorithm solving this task in time $O(N^{0.585} \cdot 1.3334^n)$.

This lower bound algorithm will in fact be used in the proof of Theorem 1.1 and relies on the above observation that we can simply use a SAT algorithm for enumerating all solutions provided the input formula has only few. The time bounds mentioned thus arise from the SAT algorithms used – the randomized 3-SAT algorithm by Hertli [9] and the deterministic one of Moser and Scheder [15] (which is in fact a derandomized version of Schöning's algorithm).

To obtain upper bounds, on the other hand, we cannot use the high-solution part of Theorem 1.1. But, although it might seem unreasonable to expect that this would yield a competitive running time, we can use an algorithm based on the bisection technique of Valiant and Vazirani [23]. Interestingly an algorithm based on Valiant and Vazirani's technique has been used already in the heuristic result of [8]. Their approach, however, is quite different from ours and does not have a good worst-case behavior.

By systematically augmenting the input formula F with randomly chosen GF(2)-linear constraints, the bisection technique makes it possible to approximate $\#F$ by determining satisfiability of the augmented formulas. The main difference of our approach to this classical scheme lies in the observation that it is more reasonable for our purposes to work directly with the system of linear equations obtained, instead of encoding it into k-CNF. In this way we obtain the running time bounds which are valid even for general CNF input formulas.

▶ Theorem 1.3 (Upper Bound Algorithm). There is an algorithm, which on input a CNF formula F on n variables and an integer $\mu \leq n$ takes time $O^*(2^{n-\mu})$ and performs the following with probability at least 2/3:

It outputs a number $u \geq \mu$ such that $U := 2^{u+3} \geq \#F$. If furthermore $u > \mu$ then 2^u is a 16-approximation of $\#F$.

Remark

This algorithm will actually work for upper bounding $|S|$ for any set $S \subseteq \{0,1\}^n$ with a polynomial membership test. However, as this is a trivial consequence of the proof, we consider only the case that the input is a CNF formula.

Moreover, we do not particularly focus on improving the approximation ratio mentioned in Theorem 1.3. Such an improvement is in fact unnecessary if we want to use this algorithm to design a randomized approximation scheme: We can combine the above algorithm with that of Theorem 1.2 to obtain a 16-approximation algorithm for e.g. #3-SAT which runs (up to a polynomial factor) within the same time bound as that stated in Theorem 1.1. This algorithm can then be plugged into a Markov chain by Jerrum and Sinclair [21] to boost the quality of approximation. This yields a $(1 + \frac{1}{\mathrm{poly}(n)})$-approximation algorithm incurring only a polynomial overhead in the computation. Thus we have a second, although more complicated, algorithm that satisfies the claim of Theorem 1.1.

2 Preliminaries

For a CNF formula F, let $\mathrm{sat}(F)$ be the set of its solutions and $\#F = |\mathrm{sat}(F)|$. We shall always use n to denote the number of variables of a CNF formula under consideration. A *randomized α-approximation algorithm* $\mathbb{A}$ for $\#k$-SAT outputs, on input a k-CNF formula F, a number $\mathbb{A}(F)$ such that

$$\Pr\left[\alpha^{-1}\#F \leq \mathbb{A}(F) \leq \alpha\#F\right] \geq p. \tag{1}$$

Here p is some constant, independent of the input and strictly larger than[2] $1/2$. A *randomized approximation scheme* for $\#k$-SAT, is then an algorithm which on input F and a natural number ϵ^{-1} behaves like a randomized e^{ϵ}-approximation algorithm.

We use the notation $x^1 = x$ and $x^0 = \bar{x}$. For a clause C, a variable x, and a truth value $a \in \{0,1\}$, the *restriction* of C on $x = a$ is the constant $\mathbf{1}$ if the literal x^a belongs to C, and $C \setminus \{x^{1-a}\}$ otherwise. We write $C|_{x=a}$ for the restriction of C on $x = a$. A *partial assignment* is a sequence of assignments $(x_1 = a_1, \ldots, x_r = a_r)$ with all variables distinct. Let α be a partial assignment. We will use the notation $\alpha \cup (x = a)$ to denote the assignment $(x_1 = a_1, \ldots, x_r = a_r, x = a)$. If C is a clause, we let $C|_\alpha$ be the result of applying the restrictions $x_1 = a_1, \ldots, x_r = a_r$ to C. Clearly the order of application does not matter. If F is a CNF formula, we let $F|_\alpha$ denote the result of applying the restriction α to each clause in F, and removing the resulting $\mathbf{1}$'s. We call $F|_\alpha$ the *residual* formula.

As we will use the algorithm of Paturi, Pudlák, Saks, and Zane [17] and a very recent paper by Hertli [9], we need the constant

$$\mu_k = \sum_{j=1}^{\infty} \frac{1}{j(j + \frac{1}{k-1})}.$$

3 The Algorithm

We are now able to state the main result in full detail.

▶ Theorem 3.1. For $k \geq 3$, $\#k$-SAT has a randomized approximation scheme running in time[3]

$$O^*\left(\epsilon^{-2} \cdot 2^{n\left(\frac{k-1}{k-1+\mu_k}\right)}\right).$$

As already outlined, the randomized approximation scheme of Theorem 3.1 is a combination of two different algorithms. We will discuss the algorithm for the case of few solutions now. The case of many solutions will be treated afterwards in Section 3.2.

3.1 Formulas with few solutions

For formulas with few solutions we will now present an algorithm relying on a simple enumeration of solutions by using a k-SAT algorithm as a subroutine. This will also prove Theorem 1.2.

[2] In the literature, usually either the value $p = 3/4$ or a further parameter δ such that $p = 1 - \delta$ seems to be common. However, it is well-known that all of these can be translated into each other with only polynomial overhead.

[3] We use the O^* notation to suppress factors sub-exponential in n.

▶ **Lemma 1.** *Let F be a k-CNF formula on n variables and let $\mathbb{A}$ be an algorithm performing the following task in time $O^*(2^{\beta_k n})$. If F is satisfiable, with probability at least $3/4$ it outputs a solution to F. If F is unsatisfiable, it reports this correctly.*

Then, there is a algorithm, which on input F and a natural number N, takes time $O^*(2^{\beta_k n} N^{(1-\beta_k)})$, and performs the following:

- *If $\#F > N$ it reports this with probability at least $3/4$.*
- *If $\#F \leq N$ then with probability at least $3/4$ it reports this and outputs the correct value $\#F$.*

Furthermore, if the algorithm reports $\#F > N$ then this holds with certainty.

Theorem 1.2 follows directly from this lemma by using the randomized 3-SAT algorithm of Hertli [9] which has $\beta_3 = 0.3864$. For the claim about the deterministic algorithm we use the result of Moser and Scheder [15], with $\beta_3 = 0.4151$.

In the proof of the above lemma, we will use the following fact which is very easily proven.

▶ **Lemma 2.** *A rooted tree with N leaves and depth (i.e. max root to leaf-distance) n has at most $n \cdot N$ vertices in total.*

Proof of Lemma 1. Note first, that by a standard trick we can boost the success probability of $\mathbb{A}$. Assume that, as provided by the statement of the lemma, we have error probability at most $1 - p \leq 1/4$. Then the probability of erring in M independent repetitions is at most $(1-p)^M \leq e^{-pM}$. Call the boosted version of this algorithm $\mathbb{A}^*$.

We shall fix a good value for M. Below we will see that algorithm $\mathbb{A}^*$ will be queried a number $O(nN)$ of times, for some $N \leq 2^n$, each time on a formula of at most n variables. The probability that the algorithm errs in any of these queries is at most $nN \cdot e^{-pM}$. So choosing M within a constant factor of $\log nN$ (which is polynomial in n) allows us to condition on the SAT algorithm not erring in any of the $O(nN)$ queries. The probability of that latter event is close to 1. And as this is the only possible source of failure of the algorithm, we will easily achieve a success probability of $3/4$ in the end.

The algorithm

Check if F is satisfiable, using $\mathbb{A}^*$, and if so, perform the following. Inductively, construct a search tree associated with partial assignments α, such that $F|_\alpha$ is satisfiable. For a leaf in the current search tree associated with some assignment α, choose a variable x from $F|_\alpha$ and check $F|_{\alpha \cup (x=0)}$ and $F|_{\alpha \cup (x=1)}$ for satisfiability using the algorithm $\mathbb{A}^*$. For each of the satisfiable restrictions add a new child to the current leaf in the search tree. We stop the algorithm, if it has N leaves, or if it has found all of the solutions of F. Traversing this tree, e.g. in a depth first manner we can implement this procedure in polynomial space (not taking the space needs of $\mathbb{A}^*$ into account).

Time

Consider the search tree this algorithm produces. As it has at most N leaves, and depth at most n, we have (recall Lemma 2) at most nN nodes overall in the tree.

Observe first, that at each node we perform at most 2 queries to the SAT algorithm $\mathbb{A}^*$. A node of level ℓ in the tree incurs queries taking time at most $O^*(b^{n-\ell})$ for $b = 2^{\beta_k}$. Therefore, we can give an upper bound of the overall time spent in answering all queries by bounding the time spent on a completely balanced binary search tree of depth $d = \log nN$.

Let $T(d, n)$ denote the overall time spend to run the algorithm on a balanced binary search tree with d levels with an n variable formula. Then, up to a sub-exponential factor

for the time spent at each node in the tree, $T(d,n) = b^n + 2T(d-1,n-1)$. Note that $T(0,n) = 1$, and thus

$$T(d,n) = \sum_{\nu=0}^{d} 2^\nu b^{n-\nu}$$

which yields the claimed bound. ◀

3.2 Formulas with many Solutions

We use the following simple folklore algorithm which can be found e.g. in Motwani and Raghavan's book [16]. Given a CNF formula F on n variables, choose an assignment from $\{0,1\}^n$ uniformly at random. Repeat this process a number N of times and let X be the number of solutions of F among these N trials. By a simple argument (see e.g. Theorem 11.1. in [16]), if

$$N = \Omega\left(\frac{2^n}{\epsilon^2 \# F}\right)$$

then with probability at least $3/4$, the value $X \cdot \frac{2^n}{N}$ is an e^ϵ-approximation of $\#F$. Hence, we have the following

▶ **Lemma 3.** *Let F be an n variable CNF formula with at least N solutions and ϵ^{-1} a natural number. Then there is an algorithm which, in time $O^*\left(\frac{2^n}{\epsilon^2 N}\right)$ yields a randomized e^ϵ-approximation of $\#F$.*

3.3 Combining the algorithms

We shall now prove Theorem 3.1 by combining both of the above algorithms. Let F be a k-CNF formula on n variables and ϵ^{-1} a natural number. We run the algorithm of Lemma 1 with a parameter N. The exact value of N will be determined later. Note that if the algorithm reports $\#F \leq N$, it also computes $\#F$ exactly with probability at least $3/4$. Otherwise, if the algorithm reports that $\#F > N$, we know with certainty that this is the case. Hence, given that the algorithm reports the latter, the algorithm of Lemma 3 will take time $O^*\left(\frac{2^n}{\epsilon^2 N}\right)$ to yield an e^ϵ-approximation of $\#F$.

It remains to bound the running time which amounts to optimizing the cutoff parameter N. For every choice of N, the combined algorithm works in time within a sub-exponential factor of

$$\max\{2^{\beta_k n} N^{(1-\beta_k)}, \frac{2^n}{N}\}.$$

Let f be such that $\log_2 N = f \cdot n$. Then this maximum translates into $\max\{\beta_k + f(1-\beta_k), 1-f\}$. Since $(\beta_k + f(1-\beta_k)$ is increasing and $1-f$ is decreasing in f, the minimum over all f of the maximum of the two is obtained when f is chosen so as to make them equal, that is, at

$$f = \frac{1-\beta_k}{2-\beta_k}.$$

This translates into an overall running time of

$$O^*\left(2^{\frac{n}{2-\beta_k}}\right).$$

Recall that β_k determines the running time $O^*(2^{\beta_k n})$ of the subroutine consisting of a randomized k-SAT algorithm, used in the algorithm of Lemma 1. We shall have a look at these running times, now.

3.3.1 The case $k \geq 5$

The algorithm of Paturi, Pudlák, Saks and Zane [17], can be used as the subroutine randomized k-SAT algorithm, which has a running time of

$$O^*\left(2^{(1-\frac{\mu_k}{k-1})n}\right). \tag{2}$$

Hence, we have here, $\beta_k = 1 - \frac{\mu_k}{k-1}$ which yields the claimed bound.

3.3.2 The cases $k = 3$ and $k = 4$

For these values of k, several improvements over the PPSZ algorithm have been presented. The currently fastest one is that by Hertli [9], whose bounds match those of the PPSZ algorithm in the unique-SAT case. We thus also have here the corresponding bound of equation (2).

4 Upper bounds

In this section we will prove Theorem 1.3 by presenting a simple algorithm producing upper bounds on $\#F$. We will use Valiant and Vazirani's bisection technique [23] and its application to approximate counting. We will therefore consider random GF(2)-linear systems of equations of the form $Ax = b$. For some $m \leq n$ these consist of an $m \times n$ matrix A, an m dimensional vector b and a vector x representing the variables of F. The entries of A and b are chosen independently and uniformly at random from $\{0, 1\}$. As such systems give rise to a family of pairwise independent hash functions of the form $h(x) = Ax - b$, we will use the following well known

▶ Fact 4.1 (Hashing Lemma). Let F be a CNF formula, and $Ax = b$ an $m \times n$ random linear system of equations. Then

$$\Pr\left[|\{x \in \mathrm{sat}(F) \mid Ax = b\}| \notin (1-\epsilon, 1+\epsilon) \cdot 2^{-m} \cdot \#F\right] \leq \frac{(1-2^{-m})2^m}{\#F \cdot \epsilon^2}.$$

The formulation of this fact in terms of CNF formulas is just for convenience. In fact, in its general wording it can be applied to any finite set – its proof can be found e.g. in Goldreich's book [6]. Secondly, we need a standard fact about the rank of random matrices such as the matrices obtained in the above way. Consider a random $m \times n$ matrix A as above with $m \leq n$ and let r denote its rank. The proof of the following lemma then follows easily, for example, from a result of Blömer, Karp and Welzl [1]:

▶ **Lemma 4.** *There is a constant c such that $\mathbb{E}[r] \geq m - c$. Furthermore, $r \geq m - O(\log m)$ with probability at least $1 - O(m^{-1})$.*

A third ingredient of the algorithm is the following Lemma, which can be proved by simple linear algebra .

▶ **Lemma 5.** *Let $Ax = b$ be a system of GF(2)-linear equations with solution set $\mathcal{A}$. There is an algorithm listing all solutions in time within a polynomial factor of $|\mathcal{A}|$.*

We are now ready to prove the Theorem.

Proof of Theorem 1.3. We start with the description of the algorithm. Choose a random GF(2)-linear $n \times n$ system of equations $Ax = b$. Starting with a parameter $\nu = n$ and decreasing ν in each step, we build random linear systems $A_n x = b_n, A_{n-1} x = b_{n-1}, \ldots, A_\mu x = b_\mu$.

The system $A_\nu x = b_\nu$ is obtained from $Ax = b$ by deleting the last $n - \nu$ rows of A and entries of b. Then F_ν denotes the pair consisting of F and $A_\nu x = b_\nu$. We say that F_ν is *satisfiable* if there is a solution $x \in \mathrm{sat}(F)$ such that $A_\nu x = b_\nu$.

For each ν we rigorously determine whether F_ν is satisfiable by using the algorithm of Lemma 5 to list all solutions of the linear system and for each determine whether it satisfies F. We let u be the minimum $\nu \geq \mu$ such that F_ν is unsatisfiable. If all $F_n, \ldots, F_\mu$ are unsatisfiable, we set $u = \mu$.

To establish the time bound, note that the running time is dominated by the time the algorithm spends in determining satisfiability of F_μ. By Lemma 4 the matrix A_μ has with high probability rank at least $m - O(\log n)$ and we have $|\mathcal{A}| = O^*(2^{n-\mu})$ which yields the claimed time bound. Thus we shall in the following condition on the event that the rank of A_μ satisfies this rank criterion.

Correctness

The correctness follows from standard arguments also used in the classical approach [23]. We give a proof for completeness. Let $f = \lceil \log \#F \rceil$. Assume first, for simplicity, that $\mu = 0$. We will show that with the desired probability, u is a 16-approximation to $\#F$.

First, consider the event that $u = f - c$ for some $c \geq 4$. The probability p_c of this event is the probability that all of $F_n, \ldots, F_u$ are unsatisfiable. Furthermore, conditional on F_u being unsatisfiable, all F_ν for $\nu \geq u$ are unsatisfiable, as well. Hence, we have $p_c = \Pr[F_u \text{ is unsat }]$, and by the Hashing Lemma 4.1, we have thus

$$\Pr[F_u \text{ is unsat }] < \frac{2^u}{\#F} \leq 2^{1-c}.$$

By a union bound argument, we thus see that $f - 3 \leq u$ with probability at least $3/4$.

Next, consider $u = f + c$ which implies that F_{u-1} is satisfiable. Applying the Hashing Lemma 4.1 with parameter $\epsilon = \zeta - 1$ for $\zeta = 2^{u-1}/\#F$, we see that

$$\Pr[F_{u-1} \text{ is satisfiable}] < \frac{\zeta}{(\zeta - 1)^2}.$$

As $\zeta \geq 1$, this bound is decreasing in ζ, and hence in u, therefore, the probability that F_{u-1} is satisfiable is at most $2^{c-1}(2^{c-1} - 1)^{-2}$. Again, a union bound shows that $u \leq f + 3$ with probability at least $33/49$.

Next, note that if $\mu > 0$ then these findings do not change. Especially, if $\mu < f - 3$ then the above result does not change. And if $\mu \geq f - 3$, then by the above, with probability at least $3/4$ we have $u = \mu$.

Taking into account that we have conditioned on A_μ having rank $m - O(\log m)$ the above probabilities degenerate a bit. But the bounds claimed in the statement of the Theorem are still easily achieved. ◀

Remarks

Note that the use of listing algorithm of Lemma 5 can be avoided by using a uniform sampling algorithm for the solutions of $Ax = b$, this then yields essentially the same time bounds. Furthermore, uniform sampling is easily achieved by fixing a basis of the column space of A, choosing u.a.r. assignments to non-basis variables and extending these assignments to solutions of $Ax = b$.

5 Open Problems

It is a peculiar fact that our result falls short of yielding any reasonable time bound for the approximation of #2-SAT. A direct application of the algorithm of Theorem 3.1 to this case would yield (using a polynomial time 2-SAT subroutine) a bound of $O(1.4142^n)$ whereas the fastest exact method [24] takes time only $O(1.2377^n)$. It would therefore be interesting to develop an approximation algorithm which beats the bounds of these exact methods.

Secondly, the time bounds achieved in this paper are significantly better than those for known exact methods, but also, they are much worse than the bounds known for the corresponding satisfiability problems. Is it possible to close this gap, maybe even in terms of a purely algorithmic analog of Valiant and Vazirani's result?

6 Acknowledgments

I would like to thank Martin Grohe for bringing this problem to my attention and for several helpful discussions on the topic. For further discussions, I would also like to thank Holger Dell, Alistair Sinclair, and Piyush Srivastava.

References

1 Johannes Blömer, Richard M. Karp, and Emo Welzl. The rank of sparse random matrices over finite fields. *Random Struct. Algorithms*, 10(4):407–419, 1997.
2 Vilhelm Dahllöf, Peter Jonsson, and Magnus Wahlström. Counting models for 2sat and 3sat formulae. *Theor. Comput. Sci.*, 332(1-3):265–291, 2005.
3 Olivier Dubois. Counting the number of solutions for instances of satisfiability. *Theor. Comput. Sci.*, 81(1):49–64, 1991.
4 Martin Fürer and Shiva Prasad Kasiviswanathan. Algorithms for counting 2-satsolutions and colorings with applications. In *AAIM*, pages 47–57, 2007.
5 Vibhav Gogate and Rina Dechter. Approximate counting by sampling the backtrack-free search space. In *AAAI*, pages 198–203, 2007.
6 Oded Goldreich. *Computational complexity*. Cambridge University Press, Cambridge, 2008. A conceptual perspective.
7 Carla P. Gomes, Jörg Hoffmann, Ashish Sabharwal, and Bart Selman. From sampling to model counting. In *IJCAI*, pages 2293–2299, 2007.
8 Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting: A new strategy for obtaining good bounds. In *AAAI*, 2006.
9 Timon Hertli. 3-sat faster and simpler - unique-sat bounds for ppsz hold in general. In *FOCS*, 2011. To appear.
10 Timon Hertli, Robin A. Moser, and Dominik Scheder. Improving ppsz for 3-sat using critical variables. In *STACS*, pages 237–248, 2011.
11 Kazuo Iwama and Suguru Tamaki. Improved upper bounds for 3-sat. In J. Ian Munro, editor, *SODA*, page 328. SIAM, 2004.
12 Lukas Kroc, Ashish Sabharwal, and Bart Selman. Leveraging belief propagation, backtrack search, and statistics for model counting. *Annals OR*, 184(1):209–231, 2011.
13 Konstantin Kutzkov. New upper bound for the #3-sat problem. *Inf. Process. Lett.*, 105(1):1–5, 2007.
14 B. Monien and E. Speckenmeyer. Solving satisfiability in less than 2^n steps. *Discrete Appl. Math.*, 10(3):287–295, 1985.
15 Robin A. Moser and Dominik Scheder. A full derandomization of Schöning's k-sat algorithm. In *STOC*, pages 245–252, 2011.

16 Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge University Press, Cambridge, 1995.

17 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for -sat. *J. ACM*, 52(3):337–364, 2005.

18 Daniel Rolf. Improved bound for the PPSZ/Schöning-algorithm for 3-sat. *JSAT*, 1(2):111–122, 2006.

19 Dan Roth. On the hardness of approximate reasoning. *Artif. Intell.*, 82(1-2):273–302, 1996.

20 Uwe Schöning. A probabilistic algorithm for k-sat and constraint satisfaction problems. In *FOCS*, pages 410–414, 1999.

21 Alistair Sinclair and Mark Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Inf. Comput.*, 82(1):93–133, 1989.

22 Larry J. Stockmeyer. On approximation algorithms for #p. *SIAM J. Comput.*, 14(4):849–861, 1985.

23 Leslie G. Valiant and Vijay V. Vazirani. Np is as easy as detecting unique solutions. *Theor. Comput. Sci.*, 47(3):85–93, 1986.

24 Magnus Wahlström. A tighter bound for counting max-weight solutions to 2sat instances. In *IWPEC*, pages 202–213, 2008.

25 Wei Wei and Bart Selman. A new approach to model counting. In *SAT*, pages 324–339, 2005.

26 Wenhui Zhang. Number of models and satisfiability of sets of clauses. *Theor. Comput. Sci.*, 155(1):277–288, 1996.

Asymptotic enumeration of Minimal Automata *

Frédérique Bassino[1], Julien David[1], and Andrea Sportiello[2]

1 **LIPN, Université Paris 13, and CNRS, UMR 7030**
99, av. J.-B. Clément, 93430 Villetaneuse, France
Frederique.Bassino@lipn.univ-paris13.fr Julien.David@lipn.univ-paris13.fr

2 **Università degli Studi di Milano, Dip. di Fisica, and INFN**
Via G. Celoria 16, 20133 Milano, Italy
Andrea.Sportiello@mi.infn.it

Abstract

We determine the asymptotic proportion of minimal automata, within n-state accessible deterministic complete automata over a k-letter alphabet, with the uniform distribution over the possible transition structures, and a binomial distribution over terminal states, with arbitrary parameter b. It turns out that a fraction $\sim 1 - C(k,b)\, n^{-k+2}$ of automata are minimal, with $C(k,b)$ a function, explicitly determined, involving the solution of a transcendental equation.

1998 ACM Subject Classification F.2 Analysis of algorithms and problem complexity

Keywords and phrases minimal automata, regular languages, enumeration of random structures

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.88

1 Introduction

To any regular language, one can associate in a unique way its minimal automaton, i.e. the only accessible complete deterministic automaton recognizing the language, with a minimal number of states. Therefore the *space complexity* of a regular language can be seen as the number of states of its minimal automaton. The worst-case complexity of algorithms dealing with finite automata is usually known [27]. But the average-case analysis of algorithms requires weighted sums on the set of possible realizations, and in particular the enumeration of the objects that are handled [10]. Therefore a precise enumeration is often required for the algorithmic study of regular languages.

The enumeration of finite automata according to various criteria (with or without initial state [18], non-isomorphic [13], up to permutation of the labels of the edges [13], with a strongly connected underlying graph [21, 18, 26, 19], acyclic [22],...) has been investigated since the fifties.

In [18] Korshunov determines the asymptotic estimate of the number of accessible complete and deterministic n-state automata over a finite alphabet. His derivation, and even the formulation of the result, are quite complicated. In [4] a reformulation of Korshunov's result leads to an estimate of the number of such automata involving the Stirling number of the second kind. On the other side, in [20] a different simplification of the involved expressions is achieved, by highlighting the role of the Lagrange Inversion Formula in the analysis.

A natural question is to ask what is the fraction of minimal automata, among accessible complete and deterministic automata of a given size n and alphabet cardinality k. Nicaud

* This work was completed with the support of the ANR project MAGNUM number 2010-BLAN-0204.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 88–99

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

[25] shows that, asymptotically, half of the complete deterministic accessible automata over a unary alphabet are minimal, thus solving the question for $k = 1$.

In this paper we solve this question for a generic integer $k \geq 2$ (see Theorem 1 later on). At a slightly higher level of generality, we give a precise estimation of the asymptotic proportion of minimal automata, within n-state accessible deterministic complete automata over a k-letter alphabet, for the uniform distribution over the possible transition structures, and a binomial distribution over terminal states, with arbitrary parameter $0 < b < 1$ (the uniform case corresponding to $b = \frac{1}{2}$). Our theoretical results are in agreement with the experimental ones.

The paper is organized as follows. In Section 2 we recall some basic notions of automata theory, and we set a list of notations that will be used in the remainder of the paper. Then, we state our main theorem, and give a short and simple heuristic argument. In Section 3 we give a detailed description of the proof structure, and its subdivision into separate lemmas. In Section 4 we prove in detail the most difficult lemmas, and give indications for those that are provable through standard methods. Finally, in Section 5 we discuss some of the implications of our result.

2 Statement of the result

For a given set E, $|E|$ denotes the cardinality of E. The symbol $[n]$ denotes the canonical n-element set $\{1, 2, \ldots, n\}$. Let $\mathcal{E}$ be a Boolean condition, the Iverson bracket $[\![\mathcal{E}]\!]$ is equal to 1 if $\mathcal{E} =$ true and 0 otherwise. We use $\mathbb{E}(X)$ to denote the expectation of the quantifier X, and $\mathbb{P}(\mathcal{E}) = \mathbb{E}([\![\mathcal{E}]\!])$ for the probability of the event $\mathcal{E}$. For $\{\mathcal{E}_i\}$ a collection of events, we define a shorthand for the first moment

$$\boldsymbol{m}(\{\mathcal{E}_i\}) := \sum_i \mathbb{P}(\mathcal{E}_i) = \mathbb{E}\Big(\sum_i [\![\mathcal{E}_i]\!]\Big)\,. \tag{1}$$

If $p(c)$ is the probability that exactly c events occur, we have $\boldsymbol{m}(\{\mathcal{E}_i\}) = \sum_c c\,p(c) \geq \sum_{c\geq 1} p(c) = 1 - p(0)$, i.e. $p(0) \geq 1 - \boldsymbol{m}(\{\mathcal{E}_i\})$. This elementary inequality, known as *first-moment bound*, is used repeatedly in the following. It is in fact a special case of a more general relation (named *Markov's inequality*), stating that, for x a random variable, $\mathbb{P}(|x| \geq a) \leq \mathbb{E}(|x|)/a$ (the specialisation is $x \in \mathbb{N}$ and $a = 1$).

A *finite deterministic automaton* A is a quintuple $A = (\Sigma, Q, \delta, q_0, \mathcal{T})$ where Q is a finite set of *states*, Σ is a finite set of *letters* called *alphabet*, the *transition function* δ is a mapping from $Q \times \Sigma$ to Q, $q_0 \in Q$ is the *initial state* and $\mathcal{T} \subseteq Q$ is the set of *terminal* (or *final*) states. With abuse of notations, we identify $\mathcal{T}(i) \equiv [\![i \in \mathcal{T}]\!]$.

An automaton is *complete* when its transition function is total. The transition function can be extended by morphism to all words of Σ^*: $\delta(p, \varepsilon) = p$ for any $p \in Q$ and for any $u, v \in \Sigma^*$, $\delta(p, (uv)) = \delta(\delta(p, u), v)$. A word $u \in \Sigma^*$ is *recognized* by an automaton when $\delta(q_0, u) \in \mathcal{T}$. The *language* recognized by an automaton is the set of words that it recognizes. An automaton is *accessible* when for any state $p \in Q$, there exists a word $u \in \Sigma^*$ such that $\delta(q_0, u) = p$.

We say that two states p, q are *Myhill-Nerode-equivalent* (or just *equivalent*), and write $p \sim q$, if, for all finite words u, $\mathcal{T}(\delta(p, u)) = \mathcal{T}(\delta(q, u))$ [24]. This property is easily seen to be an equivalence relation. An automaton is said to be *minimal* if all the equivalence classes are atomic, i.e. $p \not\sim q$ for all $p \neq q$. In other words, the minimal automaton A' recognizing the same language as A has set of states Q' corresponding to the set of equivalence classes of A. For a general reference on automata see e.g. [14].

At the aim of enumeration, the actual labeling of states in Q and letters in Σ is immaterial, and we can canonically assume that $Q = [n]$, $\Sigma = [k]$, and $q_0 = 1$. In this case, when there is no ambiguity on the values of n and k, we will associate an automaton A to a pair $(\mathcal{D}, \mathcal{T})$ of a transition structure and a set of terminal states. The set of complete deterministic accessible automata with n states over a k-letter alphabet is denoted $\mathcal{A}_{n,k}$.

We will determine statistical averages of quantities associated to automata $A \in \mathcal{A}_{n,k}$. This requires the definition of a measure $\mu(A)$ over $\mathcal{A}_{n,k}$. The simplest and more natural case is just the uniform measure. We generalise this measure by introducing a continuous parameter. For S a finite set, the *multi-dimensional Bernoulli distribution* of parameter b over subsets $S' \subseteq S$ is defined as $\mu_b(S') = b^{|S'|}(1-b)^{|S|-|S'|}$. The distribution associated to the quantifier $|S'|$ is thus the binomial distribution. We will consider the family of measures $\mu_b^{(n,k)}(A) = \mu_{\text{unif}}^{(n,k)}(\mathcal{D})\mu_b^{(n)}(\mathcal{T})$, with $\mu_{\text{unif}}^{(n,k)}(\mathcal{D})$ the uniform measure over the transition structures of appropriate size, and $\mu_b^{(n)}(\mathcal{T})$ the Bernoulli measure of parameter b over $Q \equiv [n]$. The uniform measure over all accessible deterministic complete automata is recovered setting $b = \frac{1}{2}$. Superscripts will be omitted when clear.

The result we aim to prove in this paper is

▶ **Theorem 1.** *In the set $\mathcal{A}_{n,k}$, with the uniform measure, the asymptotic fraction of minimal automata is*

$$\exp\left(-\tfrac{1}{2}c_k n^{-k+2}\right), \tag{2}$$

with

$$c_k = \tfrac{1}{2}\,{\omega_k}^k; \qquad -k\,\omega_k = \ln(1-\omega_k)\,. \tag{3}$$

More generally, for any $0 < b < 1$, with measure $\mu_b^{(n,k)}(A)$, the asymptotic fraction is

$$\exp\left(-(1-2b(1-b))c_k n^{-k+2}\right). \tag{4}$$

We singled out the constant ω_k, instead of only c_k, because the former appears repeatedly, in the evaluation of several statistical properties of random automata. Solving (3), it can be written in terms of (a branch of) the Lambert W-function, as $\omega_k = 1 + \frac{1}{k}W(-ke^{-k})$, however the implicit definition (3) is of more practical use. See Table 1 for a numerical table of values.

k	2	3	4	5	6
ω_k	0.796812	0.940480	0.980173	0.993023	0.997484
c_k	0.317455	0.415928	0.461509	0.482799	0.492498

Table 1 The constants involved in the statement of Theorem 1, for the first values of k.

The result above, specialised to $b = 1/2$, provides as a corollary the asymptotic number of minimal automata, when combined with the known asymptotics for $|\mathcal{A}_{n,k}|$ [18, 4, 20]

$$|\mathcal{A}_{n,k}| = \omega_k \left\{ {kn+1 \atop n} \right\} 2^n\big(1 + \mathcal{O}(n^{-1})\big)\,, \tag{5}$$

where $\left\{ {n \atop m} \right\}$ denotes a Stirling number of second type, i.e., the number of ways of partitioning n elements into m non-empty blocks, and ω_k is defined as in (3) (the asymptotics of Stirling numbers in this regime is then extracted through a saddle-point analysis [11]).

When it is understood that $|\Sigma| = k$, a transition function δ is identified with a k-tuple of maps (or, for short, a *k-map*) $\delta_\alpha : Q \to Q$, as $\delta_\alpha(p) \equiv \Delta(p, \alpha)$ (in this case, to avoid confusion, we use Δ for the k-tuple of $\{\delta_\alpha\}_{1\le\alpha\le k}$). And, clearly, a k-map is identified with the corresponding vertex-labeled, edge-coloured digraph over n vertices, with uniform out-degree k, such that, for each vertex $i \in [n]$ and each colour $\alpha \in [k]$, there exists exactly one edge of colour α outgoing from i. The terminology of graph theory will occasionally be used in the following.

We use the word *motif* for an unlabeled oriented graph M, when it is intended as denoting the class of (edge-coloured) subgraphs of a k-map that are isomorphic to M. The core of our proof is in the analysis of the probability of occurrence of certain motifs, that we now introduce.

▶ **Definition 2.** A *M-motif* M of a transition structure $\mathcal{D}$ is a pair of states $i \neq j$, and an ordered k-tuple of states $\{\ell_\alpha\}_{1\le\alpha\le k}$, such that $\delta_\alpha(i) = \delta_\alpha(j) = \ell_\alpha$ (see Figure 1, left). Repetitions among ℓ_α's are allowed.

A *three-state M-motif* $M^{(3)}$ of a transition structure $\mathcal{D}$ is the analogue of a M-motif, with three distinct states i, j and h, such that $\delta_\alpha(i) = \delta_\alpha(j) = \delta_\alpha(h) = \ell_\alpha$ for all $1 \le \alpha \le k$ (see Figure 1, right).

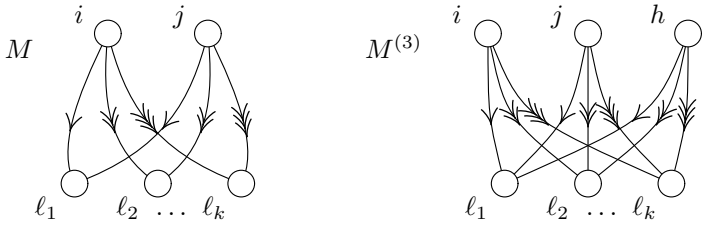


Figure 1 Left: a M-motif. Right: a three-state M-motif. The colouring of the edges is represented through the multiplicity of arrows. The examples are for $k = 3$.

The reason for studying M-motifs is in the two following easy remarks:

▶ Remark. If the transition structure of an automaton A contains a M-motif, with states i, j and $\{\ell_\alpha\}_{1\le\alpha\le k}$, and $\mathcal{T}(i) = \mathcal{T}(j)$, then $i \sim j$ and A is not minimal.

▶ Remark. Consider a transition structure $\mathcal{D}$ containing no three-state M-motifs, and r M-motifs with states $\left\{i^a, j^a, \{\ell^a_\alpha\}_{1\le\alpha\le k}\right\}_{1\le a\le r}$. Averaging over the possible sets of terminal states with the measure $\mu_b(\mathcal{T})$, the probability that $\mathcal{T}(i^a) = \mathcal{T}(j^a)$ for some $1 \le a \le r$ is $1 - (2b(1-b))^r$.

Our theorem results as a consequence of a number of statistical facts, on the structure of random automata, which are easy to believe although hard to prove. Thus, there is a short, non-rigorous path leading to the theorem, that we now explain.

1. A fraction $1-o(1)$ of non-minimal automata contain two Myhill-Nerode-equivalent states $i \sim j$ that are the incoming states of a M-motif.
2. Random transition structures locally "look like" random k-maps – this despite the highly non-local, and non-trivial, accessibility condition – the only remarkable difference being in the distribution of the incoming degrees r of the states, $p_r = 0$ if $r = 0$, and $\frac{1}{\omega_k}\mathrm{Poiss}_{k\omega_k}(r)$ if $r \ge 1$.

3. With this in mind, it is easy to calculate that the average number of M-motifs with equivalent incoming states is $(1-2b(1-b))\binom{n}{2}n^{-k}\left[\frac{\mathbb{E}(r(r-1)p_r)}{k^2}\right]^k$, at leading order in n, that is, $\frac{1}{2}(1-2b(1-b))\,{\omega_k}^k\,n^{-k+2}$.
4. Random transition structures also show weak correlations between distant parts, and M-motifs are 'small', thus, with high probability, pairs of M-motifs are non-overlapping. This suggests that the distribution of the number of M-motifs is a Poissonian, with the average calculated above (as if the corresponding events were decorrelated). As a corollary, we get the probability that there are no M-motifs. By the first claim, on the dominant role of M-motifs, this allows to conclude.

3 Structure of the proof

As it often happens, what seems the easiest way to get convinced of a claim is not necessarily the easiest path to produce a rigorous proof. Our proof strategy will be in fact very different from the sequence of claims collected above. As it is quite composite, in this section we will outline the subdivision of the proof into lemmas, and postpone the proofs to Section 4.

Call P_{rare} the probability, w.r.t. $\mu_b(\mathcal{D},\mathcal{T})$ above, that the transition structure contains no M-motif, and still the automaton is non-minimal. Call P_{confl} the probability that the transition structure contains some three-state M-motifs. Call $P(r)$ the probability that the transition structure contains no three-state M-motif, and exactly r M-motifs. Thus $P_{\text{confl}}+\sum_{r\geq 0}P(r)=1$.

The fraction of pairs $(\mathcal{D},\mathcal{T})$ of transition structures $\mathcal{D}$ with no three-state M-motif, and sets of terminal states $\mathcal{T}$ taken with the Bernoulli measure of parameter b such that $\mathcal{T}(i^a)=\mathcal{T}(j^a)$ for some M-motif, are $\sum_{r\geq 1}P(r)\,(1-(2b(1-b))^r)$. As a consequence, w.r.t. the measure $\mu_b(A)$ above, the probability that an automaton is non-minimal is

$$\mathbb{P}(A \text{ is non-minimal}) = \sum_{r\geq 1} P(r)\,(1-(2b(1-b))^r) + \mathcal{O}(P_{\text{rare}}) + \mathcal{O}(P_{\text{confl}})\,. \tag{6}$$

(To be precise, above we are neglecting summands $P^{(r)}_{\text{rare}}$, for $r\geq 1$, describing the probability of having no three-state M-motif, and exactly r M-motifs, all of which satisfying $\mathcal{T}(i^a)\neq\mathcal{T}(j^a)$, and still there exist pairs of equivalent states. From the treatment of the following section, it would be easily seen that these terms are negligible.)

If one can prove that $P_{\text{rare}}, P_{\text{confl}} = o\big(1-P(0)\big)$, then

$$\mathbb{P}(A \text{ is non-minimal}) = \sum_{r\geq 1} P(r)\big(1-(2b(1-b))^r + o(1)\big)\,. \tag{7}$$

In particular, if we can prove that $P(r)=\text{Poiss}_\rho(r)(1+o(1))$, with $\rho=\sum_r rP(r)$, it would follow that

$$\mathbb{P}(A \text{ is non-minimal}) = \big(1-e^{-\rho(1-2b(1-b))}\big)(1+o(1))\,. \tag{8}$$

This corresponds to the statement of Theorem 1, with $\rho=c_k n^{-k+2}$.

Note that our error term is not only small w.r.t. 1: as important for probabilities, it is small also w.r.t. $\min(p,1-p)$, with p the probability of our event of interest. As, for an alphabet with k letters, $p\sim n^{-k+2}$ has a non-trivial scaling with size when $k>2$, this difference is relevant.

So we see that Theorem 1 is implied by

▶ Proposition 1. The statements in the following list do hold

1. $P(r) = \mathrm{Poiss}_\rho(r)(1+o(1))$, for some ρ;
2. $\rho = c_k n^{-k+2}(1+o(1))$;
3. $P_{\mathrm{confl}} = o(n^{-k+2})$;
4. $P_{\mathrm{rare}} = o(n^{-k+2})$.

A collection of related, more explicit probabilistic statements is the following

▶ Proposition 2. The average number of occurrences of M-motifs M and three-state M-motifs $M^{(3)}$ in uniform random transition structures are respectively given by

$$\boldsymbol{m}(M) = \frac{1}{2} n^{-k+2} {\omega_k}^k \left(1+o(1)\right); \tag{9}$$

$$\boldsymbol{m}(M^{(3)}) = \frac{1}{6} n^{-2k+3} {\omega_k}^{2k} \left(1+o(1)\right). \tag{10}$$

Given that there are no three-state M-motifs, the average number of r-tuples $(M_1, \dots, M_r)$ of distinct M-motifs is given by

$$\frac{1}{r!}\boldsymbol{m}\big((M_1,\dots,M_r)\big) = \frac{1}{r!}\left(\frac{1}{2} n^{-k+2} {\omega_k}^k \left(1+o(1)\right)\right)^r. \tag{11}$$

The proof of this proposition is postponed to the end of Section 4.

Equation (9) proves $\rho = c_k n^{-k+2}\left(1+o(1)\right)$, that is, Part 2 of Proposition 1. Using the first-moment bound, equation (10) proves $P_{\mathrm{confl}} = \mathcal{O}(n^{-k+1})$ as required for Part 3 of Proposition 1.

The result in equation (11) concerning higher moments of M-motifs implies the proof of convergence of $P(r)$ to a Poissonian distribution, Part 1 of Proposition 1. The idea behind this claim is the fact that the occurrence of a M-motif with given states $\{i, j\}$ (and any k-tuple $\{\ell_\alpha\}$) is a 'rare' event, as it has a probability $\sim n^{-k}$, and, as the motifs are 'small' subgraphs, involving $\mathcal{O}(1)$ vertices, and parts of the transition structure $\mathcal{D}$ far away from each other (in the sense of distance on the graph) are weakly correlated, we expect the "Poisson Paradigm" to apply in this case, as discussed, for example, in Alon and Spencer [1, ch. 8]. A rigorous proof of this phenomenon can be achieved using the strategy called *Brun's sieve* (see e.g. [1, sec. 8.3]). The verification of the hypotheses discussed in the mentioned reference is exactly the statement of equation (11).

Thus, assuming Proposition 2, there is a single missing item in our 'checklist', namely, Part 4 of Proposition 1. We need to determine that $P_{\mathrm{rare}} = o(n^{-k+2})$. The idea behind this is that, in absence of M-motifs, there is a high probability that, for all pairs of states (i, j), certain isomorphic subgraphs of the two breadth-first search trees started from i and j visit a large number of states which are all distinct. For $i \sim j$, we need in particular that, for all the pairs of homologous states in these subgraphs, they are either both or none terminal states. The fact that they are all distinct implies that the probability for this to occur is a product of factors $1 - 2b(1-b)$, one for each such pair, thus we can concentrate on the estimation of the size of these subgraphs.

For the bounds that we need, it would suffice to produce isomorphic subgraphs of size of order $\frac{\ln n}{-\ln(1-2b(1-b))}$, but it will turn out that the largest possible subgraph is provably of size at least of order $n^{\frac{1}{4(k+1)}}$, and in fact conjecturally $\mathcal{O}(n)$.

Note that we need only an upper bound on P_{rare} (and no lower bound), and we have some freedom in producing bounds, as, at a heuristic level, we expect $P_{\mathrm{rare}} = \mathcal{O}(n^{-k+1}) \ll o(n^{-k+2})$. Our proof strategy will exploit this fact, and the following property of accessible transition functions (see [7]): given a random k-map $\Delta = \{\delta_\alpha(i)\}_{1\le i\le n, 1\le\alpha\le k}$, the number of states accessible from state 1 is a random variable $m = m(n,k)$, with average $\Theta(n)$ and

probability around the modal value[1] of order $n^{-\frac{1}{2}}$. Remarkably, if the accessible part has size m, then the induced transition structure is sampled uniformly among all transition structures of size m.

This has a direct simple consequence: if the average number of occurrences of a family of events on a random k-map is $\boldsymbol{m}(\{\mathcal{E}_i\})_{k\text{-maps}} = \mathcal{O}(n^{-\gamma})$, then the same average over random accessible transition functions of fixed size is bounded as $\boldsymbol{m}(\{\mathcal{E}_i\})_{\text{acc.}} \leq \mathcal{O}(n^{-\gamma+\frac{1}{2}})$. Actually, this bound is very generous and, if needed (but this is not our case), the extra exponent $\frac{1}{2}$ could be replaced by any $\epsilon > 0$ with some extra effort.

Thus, instead of proving that $P_{\text{rare}} = o(n^{-k+2})$, we will define the quantity P'_{rare}, exactly as P_{rare} but on random k-maps over n states. Note that the definitions of P_{rare} and P'_{rare} are based on two notions: not containing certain motifs, and not presenting pairs of Myhill-Nerode-equivalent states, and that both this notions are not confined to accessible automata, but are well-defined also for maps which are not accessible. Then we will prove that

▶ Proposition 3. $P'_{\text{rare}} = o(n^{-k+\frac{3}{2}})$.

In summary, as this proposition implies Part 4 of Proposition 1, Proposition 2 implies Parts 1 to 3 of Proposition 1, and Proposition 1 implies our main Theorem 1, we need to provide proofs of Propositions 2 and 3. This task is fulfilled in the following sections.

4 Proofs of the lemmas

Proof of Proposition 3. In a k-map, we say that a state i is a *sink state* if $\delta_\alpha(i) = i$ for all α. We say that two states $\{i, j\}$ form a *sink pair* if the set

$$N_{ij} = \{i, j, \delta_1(i), \delta_1(j), \cdots, \delta_k(i), \delta_k(j)\}$$

has cardinality $k+1$ or smaller. As easily seen through the first-moment bound, the probability of having any sink state or sink pair in a random k-map is at most of order n^{-k+1} (precisely, the overall constant is bounded by $1 + \frac{(k+1)^{2k}}{2(k-1)!}$). So, to prove that $P'_{\text{rare}} = o(n^{-k+\frac{3}{2}})$, it is enough to prove the same statement conditioned on the property that the k-map does not contain any sink state or pair.

We say that two states $\{i, j\}$ form a *quasi-sink pair* if the set N_{ij} has cardinality $k+2$. The average number of quasi-sink pairs in a random k-map is of order n^{-k+2}, thus this case must be analysed at our level of accuracy.

There exist three families of quasi-sink pairs: one family corresponds to pairs producing a M-motif; another family, that we call of *type-1*, corresponds to pairs for which there exists a value α such that $\{i, j, \delta_\alpha(i), \delta_\alpha(j)\}$ are all distinct; and a further family, that we call of *type-2*, corresponds to pairs for which any letter α is such that $\delta_\alpha(i)$ or $\delta_\alpha(j)$ is not repeated within N_{ij} (say that the first case occurs for h letters, and the second one for the remaining $k-h$ ones).

In evaluating P'_{rare}, we have excluded the M-motif case, and we are left only with type-1 and type-2 quasi-sinks. Furthermore, we have excluded sink states, so in type-2 quasi-sinks we must have both h and $k-h$ non-zero.

For a type-1 quasi-sink $\{i, j\}$, define the *pair following* $\{i, j\}$ as the pair $\{i', j'\}$ such that $i' = \delta_\alpha(i)$, $j' = \delta_\alpha(j)$, for α the first lexicographic letter such that $\{i, j, \delta_\alpha(i), \delta_\alpha(j)\}$ are all distinct. For a type-2 quasi-sink $\{i, j\}$ define the *pair following* $\{i, j\}$ as the pair $\{i', j'\}$ with $i' = \delta_1(i)$, $j' = \delta_1(j)$. Again, by first-moment estimate, the probability that there exists

[1] I.e., the most probable value.

a quasi-sink pair $\{i,j\}$, such that also the pair following it is a quasi-sink, is bounded by $\mathcal{O}(n^{-k+1})$ (for which we need that $h(k-h)>0$ in a type-2 quasi-sink), and we can further condition our k-map not to contain such motifs. If $\{i,j\}$ is a quasi-sink pair, a necessary condition for $i\sim j$ is that also $i'\sim j'$. Thus, we can bound P'_{rare} by the probability that there exist no non–quasi-sink pairs in the k-map. This is the formulation of the problem that we ultimately address.

Consider a non–quasi-sink pair $\{i,j\}$, and construct the lexicographic breadth-first tree exploration, simultaneously on the two states i and j, neglecting those branches in which, in one or both of the two trees, there is a state already visited by the exploration (call *leaves* these nodes).

Call $(v_1,v_2,\dots)$ the ordered sequence of steps in the breadth-first search, at which a leaf node is visited. For fixed integers v and h, we want to determine the probability of the event $v_h\le v$, conditioned to the event that the list has at least h items. By standard estimate of factorials, and crucially making use of the exclusion of sink and quasi-sink motifs, it can be proved for this quantity

$$\mathbb{P}(v_h\le v)\le\frac{1}{h!}\left(\frac{v(v+1)}{n-2v}\right)^h. \tag{12}$$

Set $h=k+1$. By definition, in a non–quasi-sink pair, we certainly have at least $k+1$ entries v_j. If $v=\mathcal{O}(n^\gamma)$ for some $0<\gamma<1$, we have that for each non–quasi-sink pair $\{i,j\}$

$$\mathbb{P}(v_{k+1}^{(ij)}\le v)\le\mathcal{O}(n^{-(k+1)(1-2\gamma)}). \tag{13}$$

The number of non–quasi-sink pairs is bounded by $\binom{n}{2}$, thus by first-moment bound

$$\mathbb{P}(v_{k+1}^{(ij)}\le v\text{ for all }\{i,j\})\le\mathcal{O}(n^{-k+1+2\gamma(k+1)}). \tag{14}$$

For $\gamma<\frac{1}{4(k+1)}$ we thus get $\mathbb{P}(v_{k+1}^{(ij)}\le v\text{ for all }\{i,j\})\le o(n^{-k+\frac{3}{2}})$ as needed. Thus, we know that, with probability larger than $1-o(n^{-k+\frac{3}{2}})$, all the non–quasi-sink pairs in our k-map have $v_{k+1}\gtrsim n^\gamma$, for any $\gamma<\frac{1}{4(k+1)}$. This means that, if we truncate the breadth-first search tree exploration to a depth of order $\gamma\frac{\ln n}{\ln k}$, we have at most k leaves in the tree. Thus, for all the trees, we have at least order n^γ internal nodes, i.e. pairs of states (i',j') for which it is required $\mathcal{T}(i')=\mathcal{T}(j')$ for having $i\sim j$.

But, as all these states appear not repeated in the exploration, the probability that $i\sim j$ is bounded by an exponential of the form $(1-2b(1-b))^{n^\gamma}$, which decreases faster than any power law. The overall factor $\binom{n}{2}$ from the first-moment bound is irrelevant, and we are able to conclude that $P'_{\text{rare}}=o(n^{-k+\frac{3}{2}})$, as needed. Note that this proof works not only for finite values of b in the open interval $]0,1[$, as required for our purposes, but even up to $b\gg n^{-\gamma}$. ◀

Before passing to the proof of Proposition 2, we need to recall the relation between accessible deterministic complete automata and combinatorial objects known as *k-Dyck tableaux* [4], and determine a collection of statistical properties of these tableaux.

Given the integers M and n, a *tableau* T in the set $\mathcal{T}[M\times n]$ is a map from $[M]$ to $[n]$ such that:

1. every value $y\in[n]$ has at least one preimage;
2. calling $x_T(y)$ the smallest preimage, we have $x_T(1)<x_T(2)<\ldots<x_T(n)$.

The tableau T may be represented graphically, on a $M\times n$ grid, by marking the M pairs $\{(x,T(x))\}_{1\le x\le M}$. Then the conditions above translate as follows. There is exactly one

marked entry per column. Mark in red the pairs $(x_T(y), y)$, and in black the remaining ones: there is exactly one red entry per row, which is at the left of all black entries in the same row (if any), and the polygonal line connecting the red entries in sequence is monotonically increasing. We call the collections of positions of red and black marks respectively the *backbone* B_T and *wiring part* W_T of the tableau T. It is easily seen that the number of tableaux in $\mathcal{T}[M \times n]$ is given by the Stirling number of second type $\left\{ {M \atop n} \right\}$. The asymptotic evaluation of $\left\{ {M \atop n} \right\}$, for n large and $M/n = \mathcal{O}(1)$, can be done through the general methods of analytic combinatorics (see e.g. [10], and in particular [11] for this specific problem). A result of this calculation that we shall need is the following

▶ Proposition 4. If $M, M' = \kappa n + \mathcal{O}(1)$, with $\kappa > 1$, calling ω the only solution of the equation $-\kappa\omega = \ln(1-\omega)$ in $[0,1]$,

$$\left\{ {M \atop n} \right\} = \left\{ {M' \atop n} \right\} \left(\frac{n}{\omega}\right)^{M-M'} \left(1 + o(1)\right). \tag{15}$$

For a fixed integer k, when $M = N(n,k) = kn+1$, we have a special subfamily of tableaux in $\mathcal{T}[N \times n]$. A tableau is *k-Dyck* if $x_T(\ell) \leq k(\ell-1)+1$, i.e. if the backbone cells lie above the line of slope $1/k$ containing the origin of the grid. A small example of k-Dyck tableau is shown in Figure 2.

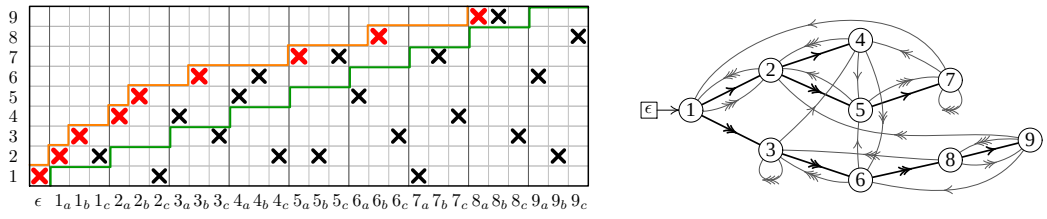


Figure 2 Left: a tableau with $n = 9$ and $k = 3$. The backbone part is in red. This tableau is valid because the red entries are monotonic (as shown by the orange profile), and k-Dyck because they are all on the left of the green staircase line. Right: the associated k-map. Backbone edges, corresponding to the breadth-first search tree, are in black, and wiring edges are in gray.

There exists a canonical bijection between k-Dyck tableaux and transition structures $\mathcal{D}$ of accessible deterministic complete automata. It suffices to associate the indices $(1, 2, \ldots, n)$ of the states to the rows of the tableaux, and the indices $(\epsilon, 1_1, \ldots, 1_k, \cdots, n_1, \ldots, n_k)$ of the oriented edges of $\mathcal{D}$ to the columns. Then, for $x = i_\alpha$, the entry (x, y) is marked in T if and only if $\delta_\alpha(i) = y$, and it is part of the backbone if and only if it is part of the breadth-first search tree on $\mathcal{D}$ started at the initial state.

Given a function $\hat{f}(y) : [n] \to [M]$, consider the restriction of the set $\mathcal{T}[M \times n]$ to tableaux T in which the backbone function $x_T(y)$ is dominated by $\hat{f}$, i.e., such that $x_T(y) \leq \hat{f}(y)$ for all $1 \leq y \leq n$. Call $\mathcal{T}[M \times n; \hat{f}]$ this set. Our k-Dyck tableaux correspond to the special case $\mathcal{T}[N \times n; \hat{f}^{\varnothing}]$, with $\hat{f}^{\varnothing}(y) := N - k(n - y + 1)$. A required technical lemma, that we state without proof, is the following

▶ Proposition 5. Take an integer n, $N = \mathcal{O}(n)$, $B = \mathcal{O}(1)$, and $\ell \gg \sqrt{n}$. Let $M = N - B$, and take a function $\hat{f}$ such that $\hat{f}(y) = \hat{f}^{\varnothing}(y)$ for all $y \leq \ell$, $\hat{f}(y) = \hat{f}^{\varnothing}(y) - B$ for all

$y \geq n-\ell$, and $\hat{f}^{\varnothing}(y) - B \leq \hat{f}(y) \leq \hat{f}^{\varnothing}(y)$ for all y. Then

$$\frac{\left|cT[M\times n;\hat{f}]\right|}{\left|cT[M\times n]\right|} - \frac{\left|cT[N\times n;\hat{f}^{\varnothing}]\right|}{\left|cT[N\times n]\right|} = o(1)\,. \tag{16}$$

With these tools at hand, we are now ready to prove Proposition 2.

Proof of Proposition 2 (p.93). Given three distinct states i, j, h, with $i<j<h$, call $\mathcal{M}_{ijh}(T)$ the event that in the tableau T there is a three-state motif on states $\{i,j,h\}$ and $\{\ell_\alpha\}$, for some ℓ_α's. Similarly, given $2r$ distinct states $\{(i_a,j_a)\}_{1\leq a\leq r}$, with $i_a<j_a$ and $j_a<j_{a+1}$, call $\mathcal{M}_{(i_1,j_1;\ldots;i_r,j_r)}(T)$ the event that in the tableau T there is a r-tuple of M-motifs, such that the a-th motif has states i_a, j_a, and $\{\ell^a_\alpha\}$, for some ℓ^a_α's. Proposition 2 consists in evaluating the two quantities

$$\sum_{i<j<h} \mathbb{E}[\![\mathcal{M}_{ijh}]\!]_{\mathcal{T}[N\times n;\hat{f}^{\varnothing}]}\,; \qquad \sum_{(i_1,j_1;\ldots;i_r,j_r)} \mathbb{E}[\![\mathcal{M}_{(i_1,j_1;\ldots;i_r,j_r)}]\!]_{\mathcal{T}[N\times n;\hat{f}^{\varnothing}]}\,. \tag{17}$$

We now make a crucial remark: given a backbone structure B, the average over all possible completions of the indicator variables $[\![\mathcal{M}_{ijh}]\!]$ (respectively $[\![\mathcal{M}_{(i_1,j_1;\ldots;i_r,j_r)}]\!]$) is zero if any column of index in the set $C=\{k(j-1)+1+\alpha, k(h-1)+1+\alpha\}_{1\leq\alpha\leq k}$ has a red mark (respectively, in the set $C=\{k(j_a-1)+1+\alpha\}_{1\leq a\leq r;1\leq\alpha\leq k}$), otherwise, it is $\prod_{i\in C} y_i^{-1}$, where y_i is the height of the backbone profile at column i. As a consequence, backbone structures contributing to the quantities in (17), weighted with the factor $\mu(\boldsymbol{c})\prod_{i\in C} y_i^{-1}$, correspond to generic backbone structures, weighted with the factor $\mu(\boldsymbol{c})$, over $(N-kr)\times n$ tableaux. The correspondence is done by just erasing the columns in C. The function $\hat{f}$ is modified accordingly. Define

$$\hat{f}^{i_1,\ldots,i_r}(y) = \hat{f}^{\varnothing}(y) - k\sum_{a=1}^{r}[\![y\geq j_a]\!]\,. \tag{18}$$

Then, the precise statement of the remark above is

$$\mathbb{E}[\![\mathcal{M}_{ijh}]\!]_{\mathcal{T}[N\times n;\hat{f}^{\varnothing}]} = \frac{\left|\mathcal{T}[(N-2k)\times n;\hat{f}^{j,h}]\right|}{\left|\mathcal{T}[N\times n;\hat{f}^{\varnothing}]\right|}\,; \tag{19}$$

$$\mathbb{E}[\![\mathcal{M}_{(i_1,j_1;\ldots;i_r,j_r)}]\!]_{\mathcal{T}[N\times n;\hat{f}^{\varnothing}]} = \frac{\left|\mathcal{T}[(N-kr)\times n;\hat{f}^{j_1,\ldots,j_r}]\right|}{\left|\mathcal{T}[N\times n;\hat{f}^{\varnothing}]\right|}\,. \tag{20}$$

Thus, the right-hand side of (19) is just the special case $r=2$ of (20). Of course we have

$$\frac{\left|\mathcal{T}[(N-kr)\times n;\hat{f}^{j_1,\ldots,j_r}]\right|}{\left|\mathcal{T}[N\times n;\hat{f}^{\varnothing}]\right|} = \frac{\frac{\left|\mathcal{T}[(N-kr)\times n;\hat{f}^{j_1,\ldots,j_r}]\right|}{\left|\mathcal{T}[(N-kr)\times n]\right|}}{\frac{\left|\mathcal{T}[N\times n;\hat{f}^{\varnothing}]\right|}{\left|\mathcal{T}[N\times n]\right|}}\,\frac{\left|\mathcal{T}[(N-kr)\times n]\right|}{\left|\mathcal{T}[N\times n]\right|}\,. \tag{21}$$

We can apply Proposition 4 to the rightmost ratio. Then, if the j_a's are within the range for application of Proposition 5, we can also simplify the leftmost ratio, to get

$$\mathbb{E}[\![\mathcal{M}_{ijh}]\!]_{\mathcal{T}[N\times n;\hat{f}^{\varnothing}]} \simeq \left(\frac{\omega_k}{n}\right)^{2k}\,; \tag{22}$$

$$\mathbb{E}[\![\mathcal{M}_{(i_1,j_1;\ldots;i_r,j_r)}]\!]_{\mathcal{T}[N\times n;\hat{f}^{\varnothing}]} \simeq \left(\frac{\omega_k}{n}\right)^{kr}\,. \tag{23}$$

As in Proposition 5 we just asked for $\ell \gg \sqrt{n}$, which is compatible with $\ell \ll n$, the fraction of $2r$-tuples $(i_1, j_1; \ldots; i_r, j_r)$ such that some j_a's are out of range is subleading, and, using the reasonings at the beginning of Section 3, the corresponding contribution can be included in P_{confl}.

Then, the straightforward calculation of the number of triplets (i, j, h), and $2r$-tuplets $\{(i_a, j_a)\}_{1 \leq a \leq r}$, at leading order in n, allows to conclude. ◀

5 Algorithmic consequences

The results obtained in this paper open new possibilities for the study in average of the properties of regular languages, and of the average-case complexity of algorithms applied to minimal automata. In this section we mention just a few among these consequences.

▶ **Corollary 3.** *Minimal automata with n states over a k-letter alphabet can be randomly generated with $\mathcal{O}(n^{3/2})$ average complexity, using Boltzmann samplers.*

The random generator for complete deterministic accessible automata given in [4] is based on a Boltzmann sampler [9], its average complexity is $\mathcal{O}(n^{3/2})$. As from Theorem 1 there is a constant proportion of minimal automata amongst accessible ones, the rejection method can be efficiently applied to randomly generate a minimal automaton. Note that such a generator (described in [4]) has already been implemented in REGAL,[2] a C++-library for the random generation of automata [2], though there were no theoretical result on the efficiency of this algorithm at that time.

▶ **Corollary 4.** *For the uniform distribution on complete deterministic accessible automata, the average complexity of Moore's state minimization algorithm is $\Theta(n \log\log n)$.*

Proof. The average complexity of Moore's state minimization algorithm for the uniform distribution on n-state deterministic automata over a finite alphabet is $\mathcal{O}(n \log\log n)$ [8]. The upper bound for accessible automata is then obtained studying the size of the accessible part of a k-random map [7, 18]. Moreover from [3] the lower bound of Moore's algorithm applied on minimal automata with n states is $\Omega(n \log\log n)$. Using Theorem 1, this is also a lower bound for complete deterministic accessible automata. ◀

▶ **Corollary 5.** *For the uniform distribution on complete deterministic accessible automata, there exists a family of implementations of Hopcroft's state minimization algorithm whose average complexity is $\mathcal{O}(n \log\log n)$.*

From [8] a family of implementations of Hopcroft's state minimization algorithm are always faster than Moore's algorithm. The result follows from Corollary 4. In [5] the lower bound on the algorithm is proved to be $\mathcal{O}(n \log n)$ for any implementation. Though it is still unknown whether there exists an implementation whose average complexity is $\Theta(n)$.

References

1 N. Alon and J. Spencer. *The Probabilistic Method.* 2nd ed., John Wiley, 2000.
2 F. Bassino, J. David and C. Nicaud. REGAL: A library to randomly and exhaustively generate automata. In J. Holub and J. Zdárek eds, *12th Int. Conference Implementation and Application of Automata (CIAA 2007)*, LNCS 4783, 303–305. Springer, 2007.

[2] Available at `http://regal.univ-mlv.fr/`

3 F. Bassino, J. David and C. Nicaud. Average-case analysis of Moore's state minimization algorithm. *Algorithmica,* to appear.

4 F. Bassino and C. Nicaud. Enumeration and random generation of accessible automata. *Theor. Comput. Sci.*, **381** 86–104, 2007.

5 J. Berstel, L. Boasson and O. Carton. Continuant polynomials and worst-case behavior of Hopcroft's minimization algorithm. *Theor. Comput. Sci.*, **410** 2811–2822, 2009.

6 J.R. Buchi. Weak second-order arithmetic and finite automata. *Math. Logic Quart.*, **6** 66–92, 1960.

7 A. Carayol and C. Nicaud. Distribution of the number of accessible states in a random deterministic automaton. In *29th International Symposium on Theoretical Aspects of Computer Science (STACS 2012)*, Paris, March 2012.

8 J. David. Average complexity of Moore's and Hopcroft's algorithms. *Theor. Comput. Sci.* to appear.

9 P. Duchon, P. Flajolet, G. Louchard and G. Schaeffer. Boltzmann Samplers for the Random Generation of Combinatorial Structures. In *Combinatorics, Probability, and Computing,* Special issue on Analysis of Algorithms **13** 577–625, 2004.

10 P. Flajolet and R. Sedgewick. *Analytic Combinatorics.* Cambridge Univ. Press, 2009.

11 I.J. Good, An Asymptotic Formula for the Differences of the Powers at Zero. *Ann. Math. Stat.* **32** 249–256, 1961.

12 F. Harary. Unsolved problems in the enumeration of graphs. *Publ. Math. Inst. Hungar. Acad. Sci.*, **5** 63–95, 1960.

13 M.A. Harrison. A census of finite automata, *Canad. Journ. of Math.*, **17** 100–113, 1965.

14 J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

15 J.E. Hopcroft. An $n \log n$ algorithm for minimizing states in a finite automaton. Technical report, Stanford CA, USA, 1971.

16 R. Iranpour and P. Chacon. *Basic Stochastic Processes: The Mark Kac Lectures.* Macmillan Publ. Co., 1988.

17 S. Kleene. Representation of Events in Nerve Nets and Finite Automata. In C. Shannon and J. Mccarthy eds., *Automata Studies*, 3–42. Princeton University Press, 1956.

18 A.D. Korshunov. Enumeration of finite automata. *Problemy Kibernetiki*, **34** 5–82, 1978. In Russian.

19 A.D. Korshunov. On the number of non-isomorphic strongly connected finite automata. *Journal of Information Processing and Cybernetics*, **9** 459–462, 1986.

20 E. Lebensztayn. On the asymptotic enumeration of accessible automata. *Discr. Math. Theor. Comp. Science* **12** 75–80, 2010

21 V.A. Liskovets. Enumeration of non-isomorphic strongly connected automata, *Vesci Akad. Navuk BSSR, Ser. Fiz.-Mat. Navuk,* **3** 26–30, 1971. In Russian.

22 V.A. Liskovets. Exact enumeration of acyclic automata. In *15-th International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC'03)*, 2003.

23 E.F. Moore. Gedanken experiments on sequential machines. In *Automata Studies,* Princeton Univ., 129–153, 1956.

24 A. Nerode. Linear automaton transformations. *Proc. of the American Math. Society*, **9** 541–544, 1958.

25 C. Nicaud. Average state complexity of operations on unary automata. In *24th Int. Symposium on Mathematical Foundations of Computer Science (MFCS 1999)*, 231–240, 1999.

26 R. Robinson, Counting strongly connected finite automata, In *Graph theory with Applications to Algorithms and Computer Science*, Y. Alavi et al. eds., Wiley, 671–685, 1985.

27 S. Yu, Q. Zhuang and K. Salomaa, The state complexities of some basic operations on regular languages, *Theoret. Comput. Sci.*, **125** 315–328, 1994.

Balanced Partitions of Trees and Applications *

Andreas Emil Feldmann[1] and Luca Foschini[2]

1 **Institute of Theoretical Computer Science, ETH Zürich**
Zürich, Switzerland
feldmann@inf.ethz.ch

2 **Department of Computer Science, U.C. Santa Barbara**
Santa Barbara, USA
foschini@cs.ucsb.edu

Abstract

We study the k-BALANCED PARTITIONING problem in which the vertices of a graph are to be partitioned into k sets of size at most $\lceil n/k \rceil$ while minimising the *cut size*, which is the number of edges connecting vertices in different sets. The problem is well studied for general graphs, for which it cannot be approximated within any factor in polynomial time. However, little is known about restricted graph classes. We show that for trees k-BALANCED PARTITIONING remains surprisingly hard. In particular, approximating the cut size is APX-hard even if the maximum degree of the tree is constant. If instead the diameter of the tree is bounded by a constant, we show that it is NP-hard to approximate the cut size within n^c, for any constant $c < 1$.

In the face of the hardness results, we show that allowing *near-balanced* solutions, in which there are at most $(1+\varepsilon)\lceil n/k \rceil$ vertices in any of the k sets, admits a PTAS for trees. Remarkably, the computed cut size is no larger than that of an optimal balanced solution. In the final section of our paper, we harness results on embedding graph metrics into tree metrics to extend our PTAS for trees to general graphs. In addition to being conceptually simpler and easier to analyse, our scheme improves the best factor known on the cut size of near-balanced solutions from $O(\log^{1.5}(n)/\varepsilon^2)$ [Andreev and Räcke TCS 2006] to $O(\log n)$, for weighted graphs. This also settles a question posed by Andreev and Räcke of whether an algorithm with approximation guarantees on the cut size independent from ε exists.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases balanced partitioning, bicriteria approximation, hardness of approximation, tree embeddings

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.100

1 Introduction

In this paper we study the k-BALANCED PARTITIONING problem, which asks for a partition of the n vertices of a graph into k sets of size at most $\lceil n/k \rceil$ each, such that the number of edges connecting vertices in different sets, called the *cut size*, is minimised. The problem has numerous applications of which one of the most prominent is in the field of parallel computing. There, it is crucial to evenly distribute n tasks (vertices) among k processors (sets) while minimising the inter-processor communication (edges between different sets), which constitutes a bottleneck. Other applications can be found in the design of electronic circuits and sparse linear solvers. However, despite the broad applicability, k-BALANCED

* The first author is supported by the Swiss National Science Foundation under grant 200021_125201/1. The second author is supported by the National Science Foundation grant IIS 0904501.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 100–111

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

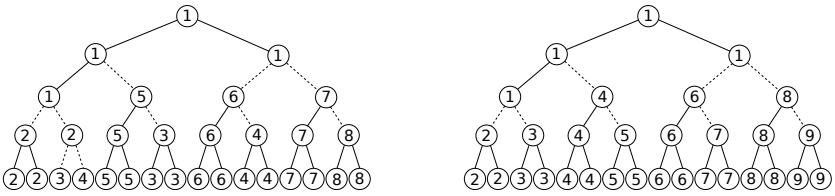


Figure 1 Two optimally partitioned binary trees. For the tree on the left $k = 8$ (with a cut size of 10) whereas $k = 9$ (with a cut size of 8) for the tree on the right. The numbers in the vertices indicate the set they belong to and the cut set is represented by the dashed edges.

PARTITIONING is a notoriously hard problem. The special case of $k = 2$, commonly known as the BISECTION problem, is already NP-hard [11]. For this reason, approximation algorithms that find a balanced partition with a cut size larger than optimal have been developed. We follow the convention of denoting the approximation ratio on the cut size by α.

Unfortunately, when k is not constant even finding an approximation of the minimum balanced cut still remains infeasible, as it is known that no finite approximation for the cut size can be computed in polynomial time, unless P=NP [1]. In order to overcome this obstacle, relaxing the balance constraints has proven beneficial. By that we mean that the sets of the partitions are allowed to be of size at most $(1 + \varepsilon)\lceil n/k \rceil$ for some factor $\varepsilon \geq 0$. Along these lines, *bicriteria approximation* algorithms have been proposed, which approximate both the balance and the cut size of the optimal solution. That is, the computed cut size is compared to the optimal cut size of a *perfectly balanced* partition in which $\varepsilon = 0$.

The k-BALANCED PARTITIONING problem has received some attention for the case $\varepsilon \geq 1$, that is when a perfect balance is approximated within a factor of two. For this case, the best result is by Krauthgamer *et al.* [15] who give an algorithm with approximation factor $\alpha \in \mathcal{O}(\sqrt{\log n \log k})$. However, it is not hard to imagine how the slack on the balance can be detrimental in practical applications. In parallel computing, for instance, a factor of two on the balance in the workload assigned to each machine can result in a factor of two slowdown, since the completion time is solely determined by the overloaded machines.

Therefore the case of $\varepsilon < 1$, that is when *near-balanced* partitions are allowed, is of greater practical interest. No progress has been made on near-balanced partitions since Andreev and Räcke [1] gave an algorithm with $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$—a significantly worse bound than the one for $\varepsilon \geq 1$. This is not surprising since, as argued in [1], as ε approaches 0 and the constraint on the balance becomes more stringent the k-BALANCED PARTITIONING problem starts bearing more resemblance to a packing problem than to a partitioning problem. One direct side effect is that the spreading metric relaxations developed for $\varepsilon \geq 1$ [5, 15] do not extend to near balanced partitions. This is because such strategies aim at breaking the graph into components of size less than n/k while minimizing the cut, only to later rely on the fact that pieces of that size can be packed into k partitions such that no partition exceeds $2n/k$ vertices. However, when near-balanced solutions are required, the partition phase can no longer be oblivious of the packing. So it is necessary to combine the partition step with an algorithm to pack the pieces into near-balanced partitions.

Our Contribution. As argued above, the restriction to near-balanced partitions poses a major challenge in understanding the structure of k-BALANCED PARTITIONING. For this reason, we consider the simplest non-trivial instance class of the problem, namely connected trees. Figure 1 gives an example of how balanced partitions exhibit a counter-intuitive behaviour even on *perfect binary trees*, as increasing k does not necessarily entail a larger cut size. Our results confirm this intuition when a perfectly balanced solution is required: adapting an argument by Andreev and Räcke [1], we show that it is NP-hard to approximate

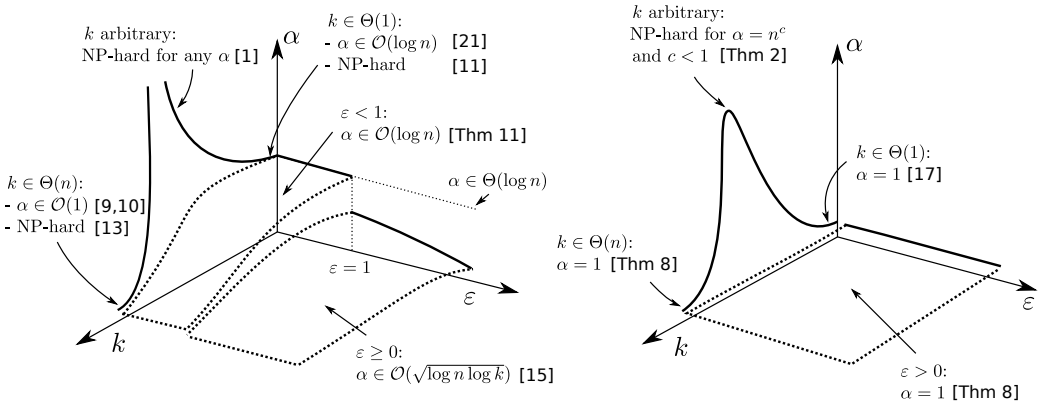


Figure 2 Illustrations of best approximation factor α known against k and ε, for general graphs (left) and trees (right). The plane (α, k) represents the case of perfectly balanced solutions ($\varepsilon = 0$) and shows that the restriction to trees does not significantly change the asymptotic behaviour. However, for $\varepsilon > 0$ much better approximations can be devised for trees. This remarkable behaviour can be partially adapted to general graphs via tree decompositions, allowing us to reduce the gap between the case $\varepsilon < 1$ and $\varepsilon \geq 1$ visible in the plot on the left.

the cut size within any factor better than $\alpha = n^c$ for any constant $c < 1$. (Figure 2 summarises the results presented here together with the related work.) This is asymptotically tight, since a trivial approximation algorithm can achieve a ratio of n by cutting all edges of the tree. Interestingly, the lower bound remains true even if the diameter of the tree (i.e. the length of the longest path between any two leaves) is restricted to be at most 4, while instances of diameter at most 3 are polynomially solvable.

By a substantially different argument, we show that a similar dichotomy arises when parametrizing the complexity with respect to the maximum degree Δ. For trees with $\Delta = 2$ (i.e. paths) k-`BALANCED PARTITIONING` is trivial. However, if $\Delta = 5$ the problem becomes NP-hard and with $\Delta = 7$ we show it is APX-hard. Finding where exactly the dichotomy arises, i.e. the $\Delta \in \{3, 4\}$ at which k-`BALANCED PARTITIONING` becomes hard, is an interesting open problem. These results should be contrasted with a greedy algorithm by MacGregor [17] to find tree bisections that can be extended to find perfectly balanced partitions for k sets with $\alpha \in \mathcal{O}(\log(n/k))$, for trees of constant degrees.

On the positive side, we show that when near-balanced solutions are allowed, trees behave substantially better than general graphs. We present an algorithm that computes a near-balanced partition for any constant $\varepsilon > 0$ in polynomial time, achieving a cut size no larger than the optimal for a perfectly balanced partition, i.e. $\alpha = 1$. In this sense the presented algorithm is a PTAS w.r.t. the balance of the computed solution. In addition, our PTAS can be shown to yield an optimal *perfectly balanced* solution for trees if $k \in \Theta(n)$, while on general graphs the problem is NP-hard for these values of k [13].

In the last section of our paper we capitalise on the PTAS for trees to tackle the k-`BALANCED PARTITIONING` problem on general, weighted graphs. By embedding a graph into a collection of trees with a cut distortion of $\mathcal{O}(\log n)$ we can use our PTAS for trees to get a solution for graphs. Since the PTAS has approximation factor $\alpha = 1$, the total approximation factor paid for the general graphs is due only to the distortion of the embedding, that is $\alpha \in \mathcal{O}(\log n)$. Note that since the graph is decomposed into trees as a preliminary step, the decomposition is oblivious of the balance constraints related to solving k-`BALANCED PARTITIONING` on the individual trees, hence the distortion does not depend on ε. This is sufficient to simultaneously improve on the previous best result known [1]

of $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$, and answers an open question posed in the same paper whether an algorithm with no dependence on ε in the ratio α exists. In addition, our analysis is significantly simpler than the one in [1]: the ad-hoc approach of [1] must deal directly with the complications introduced by considering general graphs. We are able to minimise those by relying on the powerful tool of tree decompositions.

Related Work. We extend the results in [1], where it is shown that for general graphs approximating the cut size of the k-`BALANCED PARTITIONING` problem is NP-hard for any finite factor α, if perfectly balanced partitions are needed. In [1] the authors also give a bicriteria approximation algorithm with $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$ when solutions are allowed to be near-balanced. If more unbalance is allowed, for $\varepsilon \geq 1$ Even *et al.* [5] present an algorithm with $\alpha \in \mathcal{O}(\log n)$ using spreading metrics techniques. Later Krauthgamer *et al.* [15] improved the result to $\alpha \in \mathcal{O}(\sqrt{\log n \log k})$ using a semidefinite relaxation which combines l_2 metrics with spreading metrics. To the best of our knowledge, the only result on restricted graph classes is for graphs with excluded minors (such as planar graphs). By applying a spreading metrics relaxation and the results in [14][1] it is possible to compute near-balanced solutions with $\alpha \in \mathcal{O}(1)$.

The special case when $k = 2$, commonly known as the `BISECTION` problem, is well studied. It is NP-hard in the general case [11] but approximation algorithms are known. For instance Räcke [21] gives an algorithm with approximation ratio $\alpha \in \mathcal{O}(\log n)$ for perfectly balanced partitions. For near-balanced partitions Leighton and Rao [16] show how to compute a solution using *min-ratio cuts*. In this solution the cut size is approximated within $\alpha \in \mathcal{O}(\gamma/\varepsilon)$, where γ is the approximation factor of computing a min-ratio cut. In [16] it was shown that $\gamma \in \mathcal{O}(\log n)$, and this result was improved [2] to $\gamma \in \mathcal{O}(\sqrt{\log n})$. For (unweighted) planar graphs it is possible to compute the optimum min ratio cut in polynomial time [19]. If a perfectly balanced solution to `BISECTION` is required for planar graphs, Dìaz *et al.* [4] show how to obtain a PTAS. Even though it is known that the `BISECTION` problem is weakly NP-hard on planar graphs with vertex weights [19], whether it is NP-hard on these graphs in the unweighted case is unknown. For other special graph classes the problem can be solved optimally in polynomial time [3]. For instance an $\mathcal{O}(n^4)$ algorithm for grid graphs without holes has been found [8], while for trees an $\mathcal{O}(n^2)$ algorithm [17] exists.

In addition to the case $k = 2$, some results are known for other extreme values of k. For trees the above mentioned bisection algorithm by MacGregor [17] is easily generalised to solve the k-`BALANCED PARTITIONING` problem for any constant k in polynomial time. At the other end of the spectrum, i.e. when $k \in \Theta(n)$, it is known that the problem is NP-hard [13] for any $k \leq n/3$ on general graphs. Feo and Khellaf [10] give a $\alpha = n/k$ approximation algorithm for the cut size which was improved [9] to $\alpha = 2$ in case k equals $n/3$ or $n/4$.

We complete the review of related work by discussing the literature on tree decompositions, which we leverage in our algorithm for general graphs. Informally a tree decomposition of a graph G is a set of trees for which the leaves correspond to the vertices of G, and for which the structure of their cuts approximate the cuts in G. Tree decompositions have been studied in the context of oblivious routing schemes (see [18] for a survey). In [21], Räcke introduces an optimal decomposition with factor $\mathcal{O}(\log n)$, which we employ in the present work. In a recent work, Madry [18] shows that it is possible to generalise Räcke's insights so that any *cut based* problem (see [18] for more details) is solvable on graphs by computing solutions on tree decompositions of the input graph.

[1] We thank an anonymous reviewer for pointing out this folklore result.

2 The Hardness of Computing Perfectly Balanced Partitions

We now consider the problem of finding a perfectly balanced partition of a tree with minimum cut size. We prove hardness results in the case where either the diameter or the degrees are restricted to be constant. All reductions are from MAX-3-PARTITION, defined as follows.

▶ **Definition 1** (MAX-3-PARTITION)**.** Given $3k$ integers $a_1, \ldots, a_{3k}$ and a threshold s, such that $s/4 < a_i < s/2$ and $\sum_{i=1}^{3k} a_i = ks$, find the maximum number of disjoint triples of a_1 to a_{3k} such that each triple sums up to exactly s.

The MAX-3-PARTITION problem is APX-hard, i.e. for some constant ρ it is NP-hard to decide whether all k integers or at most k/ρ of them can be partitioned into triples that sum up to exactly s. This is true even if all integers are polynomially bounded in k, which can be shown using the standard reduction for the corresponding decision problem in [11] and the results on the 3D-MATCHING problem by Petrank [20] [2].

We begin by showing that an approximation algorithm with factors $\alpha = n^c$ and $\varepsilon = 0$, for any constant $c < 1$, for k-BALANCED PARTITIONING on trees could be used to find the optimal solution to an instance of MAX-3-PARTITION. We do not rely on the APX-hardness of the latter problem for the proof, but only on the NP-hardness of the corresponding decision problem. The idea for the reduction is similar to the one used by Andreev and Räcke [1] for general graphs. The result holds even if the diameter of the tree is bounded by a constant[3].

▶ **Theorem 2.** *The k-BALANCED PARTITIONING problem on trees has no polynomial time approximation algorithm with approximation factors $\alpha = n^c$ and $\varepsilon = 0$, for any constant $c < 1$, even if the diameter is at most 4, unless P=NP.*

The reduction in the proof of the above theorem relies on the fact that the degree of the tree is unbounded. Hence a natural question arising is what the complexity of bounded degree trees is. As we will show next the problem remains surprisingly hard when bounding the degree. We are able to prove two hardness results for this case. First we show that the problem of finding a perfectly balanced partition of a tree is APX-hard even if the maximum degree of the tree is at most 7. To prove this result we use a gap-preserving reduction from MAX-3-PARTITION. In particular this means that a substantially different (and more involved) technique than the one used to prove Theorem 2 has to be employed: rather than relying on the fact that many edges have to be cut in a gadget consisting of a high degree star, we need gadgets with structural properties that guarantee a number of cut edges proportional to the number of integers that cannot be packed into triples in a MAX-3-PARTITION instance.

▶ **Theorem 3.** *Unless P=NP, there exists a constant ρ such that the minimum cut size of a perfectly balanced partition on trees with maximum degree Δ cannot be approximated in polynomial time within $\alpha = 1 + (1 - \rho^{-1})/24$ and $\varepsilon = 0$ even if Δ is at most 7.*

Proof. Given an instance of MAX-3-PARTITION with polynomially bounded integers a'_i, consider the instance I where $a_i = 12a'_i$. Obviously all hardness properties are preserved by this transformation. As a consequence all integers are divisible by 4 and $s > 20$, which will become important later in the proof. For each a_i in I, construct a gadget T_i composed by a path on a_i vertices (called a_i-path) connected to the root of a tree on s vertices (called

[2] We thank Nikhil Bansal for pointing out the connection between the reductions in [11] and the results by Petrank [20]

[3] All missing proofs of this paper can be found in the full version [7].

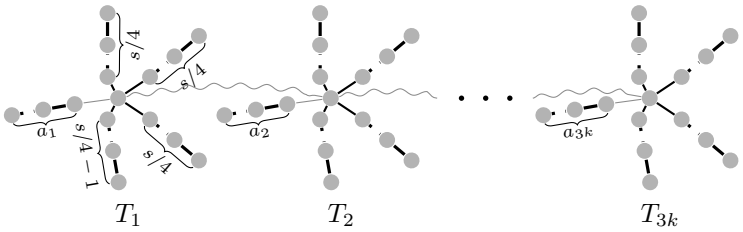


Figure 3 Construction for Theorem 3. Each gadget T_i is composed by an a_i-path connected to the root of an s-tree through an edge from A (straight grey). Each s-tree branches into four paths of (almost) the same length. Two adjacent gadgets in a path are connected through the roots of their s-trees with an edge from B (wiggled grey).

s-tree). The root of the s-tree branches into four paths, three of them with $s/4$ vertices each, and one with $s/4-1$ vertices. The construction is completed by connecting the roots of the s-trees in a path, as shown in Figure 3. We call B the set of edges connecting different T_is and A the set of edges connecting an a_i-path with the corresponding s-tree in each T_i.

At a high level, we set out to prove that if all k integers in I can be partitioned into triples that sum up to exactly s, then the constructed tree T can be split into a $4k$-balanced partition with cut size $6k-1$. If however at most a ρ fraction of the integers can be partitioned in this way, T requires at least $(1-\rho^{-1})k/4$ additional cut edges. This means that a polynomial time algorithm computing an approximate $4k$-balanced partition for T within factor $\frac{6k-1+(1-\rho^{-1})k/4}{6k-1} \geq 1+(1-\rho^{-1})/24$ of the optimum cut size could approximate the `MAX-3-PARTITION` problem within the ratio ρ in polynomial time. Hence the theorem follows.

It is easy to see that if all k integers of I can be partitioned into triples of size exactly s, cutting exactly the $6k-1$ edges in A and B suffices to create a valid $4k$-balanced partition. It remains to be shown that $(1-\rho^{-1})k/4$ additional edges are required in the other case. Let C^* be an optimal cut set of a $4k$-balanced partition in T when at most k/ρ many integers can be partitioned into triples of size exactly s in I. We argue that by incrementally repositioning cut edges from the set $C := C^* \setminus (A \cup B)$ to edges in $(A \cup B) \setminus C^*$, eventually all the edges in $A \cup B$ will be cut. However, the following lemma implies that a constant fraction of the edges initially in C will not be moved. We will then argue that the more triples of I can not be packed into triples of size s, the more edges are left in C. Thus the more edges must additionally have been in C compared to those in $A \cup B$. We rely on the following lemma.

▶ **Lemma 4.** *If $s > 20$ then $|C| \geq 2|(A \cup B) \setminus C^*|$.*

Consider the following algorithm $\mathcal{A}$ which repositions cut edges from a $4k$-balanced partition and thereby computes an approximation to `MAX-3-PARTITION`. As long as there is an uncut edge $e \in A \cup B$, $\mathcal{A}$ removes a cut edge in C and cuts e instead. At the end of the process, when all edges in $A \cup B$ are cut, $\mathcal{A}$ removes the set of cut edges left in C denoted by C'. Then $|C'|$ is the number of additional edges cut in the case at most a ρ fraction of the integers can be partitioned into triples of size exactly s. When repositioning a cut edge from C to $A \cup B$, or when removing a cut from C', $\mathcal{A}$ modifies the sizes of the sets in the partition induced by the cut set, and the balance might be lost. In particular, when a cut edge $e \in C$ is removed, the algorithm will join the two connected components induced by the cut set and incident to e to form a single component. The algorithm will then include it in an arbitrary one of the sets that contained the two components. This changes the sizes of at most two sets in the partition. When a new cut is introduced by $\mathcal{A}$, a component is split into two and the two newly created components are retained in the same set, thus no set size is changed.

By Lemma 4 there are at least as many edges in C', as there are edges that are repositioned from C to $A \cup B$. Since each edge from C repositioned by $\mathcal{A}$ causes at most two changes in set sizes, the total number of set size changes performed by $\mathcal{A}$ amortized on the removed edges in C' is therefore at most $4|C'|$.

When $\mathcal{A}$ terminates only edges from $A \cup B$ are cut. Therefore the remaining connected components correspond to the $3k$ integers a_i of I in addition to $3k$ integers equal to s. Since at most a ρ fraction of the k integers in I can be partitioned into triples of size exactly s, and the elements of size s can be ignored, at least $(1-\rho^{-1})k$ of the sets of the resulting partition do not have size exactly s. This means that $\mathcal{A}$ must have changed the size of at least $(1-\rho^{-1})k$ sets, since it converted a $4k$-balanced partition into a solution to `MAX-3-PARTITION` with at least $(1-\rho^{-1})k$ unbalanced sets. This finally implies that $4|C'| \geq (1-\rho^{-1})k$, which concludes the proof since $|C'|$ is the number of additional cuts required. ◀

Using a similar argument as in the above proof, if we restrict the degree to be at most 5 we can still show that the problem remains NP-hard. For this we use a slightly different construction than the one shown in Figure 3: instead of connecting the s-trees through their roots, the B edges connect the leaves of the shortest branches of the s-trees. It is then possible to show that exactly the $6k-1$ edges in A and B are cut if all k integers in I can be partitioned into triples of size exactly s, while otherwise at least $6k$ edges are cut. Since the `MAX-3-PARTITION` problem is NP-hard this suffices to establish the following result.

▶ **Theorem 5.** *The k-`BALANCED PARTITIONING` problem on trees has no polynomial time algorithm even if the maximum degree is at most 5, unless P=NP.*

3 Computing Near-Balanced Partitions

The previous section shows that approximating the cut size of k-`BALANCED PARTITIONING` is hard even on trees if perfectly balanced partitions are desired. We showed that for the general case when the degree is unbounded there is no hope for a polynomial time algorithm with non-trivial approximation. Therefore, in this section we study the complexity of the problem when allowing the partitions to deviate from being perfectly balanced. In contrast to the negative results presented so far, we prove the existence of a PTAS for k-`BALANCED PARTITIONING` on trees that computes near-balanced partitions but returns a cut size no larger than the optimum of a perfectly balanced partition.

Conceptually one could find a *perfectly balanced* partition of a tree T with minimum cut size in two steps. First all the possible ways of cutting T into connected components are grouped into equivalence classes based on the sizes of their components. That is, the sets of connected components $\mathcal{S}$ and $\mathcal{S}'$ belong to the same equivalence class if they contain the same number of components of size x for all $x \in \{1, \ldots, \lceil n/k \rceil\}$. In a first step the set of connected components that achieves the cut of minimum size for each class is computed and set to be the representative of the class. In a second stage only the equivalence classes whose elements can be packed into k sets of size at most $\lceil n/k \rceil$ are considered, and among those the representative of the class with minimum cut size is returned. Clearly such an algorithm finds the optimal solution to the k-`BALANCED PARTITIONING` problem, but the runtime is exponential in n as the total number of equivalence classes is exponential. The idea of our algorithm is instead to group sets of connected components into coarser equivalence classes, defined as follows. The coarser definition allows reducing the total number of classes at the expense of introducing an approximation error in the balance of the solution.

▶ **Definition 6.** Let $\mathcal{S}$ be a set of disjoint connected components of the vertices of T, and $\varepsilon > 0$. A vector $\vec{g} = (g_0, \ldots, g_t)$, where $t = \lceil \log_{1+\varepsilon}(1/\varepsilon) \rceil + 1$, is called the *signature* of $\mathcal{S}$ if in $\mathcal{S}$ there are g_0 components of size in $[1, \varepsilon\lceil n/k \rceil)$ and g_i components of size in $[(1+\varepsilon)^{i-1} \cdot \varepsilon\lceil n/k \rceil, (1+\varepsilon)^{i} \cdot \varepsilon\lceil n/k \rceil)$, for each $i \in \{1, \ldots, t\}$.

The first stage of our algorithm uses a dynamic programming scheme to find a set of connected components of minimum cut size among those with signature $\vec{g}$, for any possible $\vec{g}$. Let $\mathbb{S}$ denote the set containing each of these optimal sets that cover all vertices of the tree, as computed by the first stage. In the second stage the algorithm attempts to distribute the connected components in each set $\mathcal{S} \in \mathbb{S}$ into k bins, where each bin has a capacity of $(1+\varepsilon)\lceil n/k \rceil$ vertices. This is done using a scheme originally proposed by Hochbaum and Shmoys [12, 22] for the `BIN PACKING` problem. The final output of our algorithm is the partition of the vertices of the given tree that corresponds to a packing of a set $\widetilde{\mathcal{S}} \in \mathbb{S}$ that uses at most k bins and has minimum cut size. Both stages of the algorithm have a runtime exponential in t. Hence the runtime is polynomial if ε is a constant.

We now describe the dynamic programming scheme to compute the set of connected components of minimum cut size among those whose signature is $\vec{g}$, for every possible $\vec{g}$. We fix a root $r \in V$ among the vertices of T, and an ordering of the children of every vertex in V. We define the *leftmost* and the *rightmost* among the children of a vertex, the siblings *left of* a vertex, and the *predecessor* of a vertex among its siblings according to this order in the natural way. The idea is to recursively construct a set of disjoint connected components for every vertex $v \neq r$ by using the optimal solutions of the subtrees rooted at the children of v and the subtrees rooted at the siblings left of v. More formally, let for a vertex $v \neq r$ the set $L_v \subset V$ contain all the vertices of the subtrees rooted at those siblings of v that are left of v and at v itself (Figure 4). We refer to a set $\mathcal{F}$ of disjoint connected components as a *lower frontier of* L_v if all components in $\mathcal{F}$ are contained in L_v and the vertices in V not covered by $\mathcal{F}$ form a connected component containing the root r. For every vertex v and every signature $\vec{g}$, the algorithm recursively finds a lower frontier $\mathcal{F}$ of L_v with signature $\vec{g}$. Finally, a set of connected components with signature $\vec{g}$ covering all vertices of the tree can be computed using the solutions of the rightmost child of the root. The algorithm selects a set having minimum cut size in each recursion step. Let $C_v(\vec{g}, m)$, for any vertex $v \neq r$ and any integer m, denote the optimal cut size over those lower frontiers of L_v with signature $\vec{g}$ that cover a total of m vertices with their connected components. If no such set exists let $C_v(\vec{g}, m) = \infty$. Additionally, we define $\mu := (1+\varepsilon)\lceil n/k \rceil$, and $\vec{e}(x)$ for any integer $x < \mu$ to be the signature of a set containing only one connected component of size x. We now show that the function $C_v(\vec{g}, m)$ can be computed using a dynamic program.

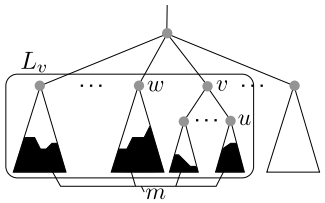


Figure 4 A part of a tree in which a vertex v, its rightmost child u, its predecessor w, the set of vertices L_v, and the m covered vertices by some lower frontier with signature $\vec{g}$ are indicated.

Let $\mathcal{F}^*$ denote an optimal lower frontier associated with $C_v(\vec{g}, m)$. First consider the case when v is a leaf and the leftmost among its siblings. Then $L_v = \{v\}$ and hence the set $\mathcal{F}^*$ either contains $\{v\}$ as a component or is empty. In the latter case the cut size is 0 and in the former it is 1 since the leaf has to be cut from the tree. Thus $C_v((0, \ldots, 0), 0) = 0$ and $C_v(\vec{e}(1), 1) = 1$ while all other function values equal infinity. Now consider the case when v is neither a leaf nor the leftmost among its siblings, and let w be the predecessor and u the rightmost child of v. The set L_v contains the vertices of the subtrees rooted at v's siblings that are left of v and at v itself. The lower frontier $\mathcal{F}^*$ can either be one in which the edge

from v to its parent is cut or not. In the latter case the m vertices that are covered by $\mathcal{F}^*$ do not contain v and hence are distributed among those in L_w and L_u since $L_v = L_w \cup L_u \cup \{v\}$. If x of the vertices in L_u are covered by $\mathcal{F}^*$ then $m - x$ must be covered by $\mathcal{F}^*$ in L_w. The vector $\vec{g}$ must be the sum of two signatures $\vec{g}_u$ and $\vec{g}_w$ such that the lower frontier of L_u (respectively L_w) has minimum cut size among those having signature $\vec{g}_u$ (respectively $\vec{g}_w$) and covering x (respectively $m - x$) vertices. If this were not the case the lower frontier in L_u (respectively L_w) could be exchanged with an according one having a smaller cut size—a contradiction to the optimality of $\mathcal{F}^*$. Hence in case v is a non-leftmost internal vertex and the edge to its parent is not cut, $C_v(\vec{g}, m)$ equals

$$\min\left\{C_w(\vec{g}_w, m - x) + C_u(\vec{g}_u, x) \mid 0 \leq x \leq m \wedge \vec{g}_w + \vec{g}_u = \vec{g}\right\}. \tag{1}$$

If the edge connecting v to its parent is cut in $\mathcal{F}^*$, then all n_v vertices in the subtree rooted at v are covered by $\mathcal{F}^*$. Hence the other $m - n_v$ vertices covered by $\mathcal{F}^*$ must be included in L_w. Let x be the size of the component $S \in \mathcal{F}^*$ that includes v. Analogous to the case before, the lower frontiers L_u and L_w with signatures $\vec{g}_u$ and $\vec{g}_w$ in $\mathcal{F}^* \setminus \{S\}$ must have minimum cut sizes. Hence the vector $\vec{g}$ must be the sum of $\vec{g}_u$, $\vec{g}_w$, and $\vec{e}(x)$. Therefore in case the edge to v's parent is cut, $C_v(\vec{g}, m)$ equals

$$1 + \min\left\{C_w(\vec{g}_w, m - n_v) + C_u(\vec{g}_u, n_v - x) \mid 1 \leq x < \mu \wedge \vec{g}_w + \vec{g}_u + \vec{e}(x) = \vec{g}\right\}. \tag{2}$$

Taking the minimum value of the formulas in (1) and (2) thus correctly computes the value for $C_v(\vec{g}, m)$ for the case in which v is neither a leaf nor the leftmost among its siblings. In the two remaining cases when v is either a leaf or a leftmost sibling, either the vertex u or w does not exist. Therefore for these cases the recursive definitions of C_v can easily be derived from formulas (1) and (2) by letting all function values $C_u(\vec{g}, x)$ and $C_w(\vec{g}, x)$ of a non-existent vertex u or w be 0 if $\vec{g} = (0, \ldots, 0)$ and $x = 0$, and ∞ otherwise.

The above recursive definitions for C_v give a framework for a dynamic program that computes the solution set $\mathbb{S}$ in polynomial time if ε is a constant, as the next theorem shows.

▶ **Theorem 7.** *For any tree and constant $\varepsilon > 0$ the set $\mathbb{S}$ is computable in polynomial time.*

The second stage of the algorithm attempts to pack each set of connected components $\mathcal{S} \in \mathbb{S}$ into k bins of capacity $(1+\varepsilon)\lceil n/k \rceil$. This means solving the well known `BIN PACKING` problem, which is NP-hard in general. However we are able to solve it in polynomial time for constant ε using a method developed by Hochbaum and Schmoys [12]. The description of the second phase has been deferred to the full version of the paper [7]. The main theorem resulting from the algorithms of the first and second phase is stated next.

▶ **Theorem 8.** *For any tree T, $k \in \{1, \ldots, n\}$, and $\varepsilon > 0$ there is an algorithm that computes a partition of T's vertices into k sets such that each set has size at most $(1+\varepsilon)\lceil n/k \rceil$ and its cut size is at most that of an optimal perfectly balanced partition of the tree. Furthermore the runtime is polynomial if ε is a constant and can be upper bounded by $\mathcal{O}(n^4(k/\varepsilon)^{1+3\lceil \frac{1}{\varepsilon}\log(\frac{1}{\varepsilon})\rceil})$.*

4 Extension to Unrestricted Weighted Graphs

In this section we present an algorithm that employs the PTAS given in Section 3 to find a near-balanced partition of a graph G with weighted edges. The (weighted) cut size computed has a capacity of at most $\alpha \in \mathcal{O}(\log n)$ times that of an optimal perfectly balanced partition of G. The algorithm relies on using our PTAS to compute near-balanced partitions of a set of decomposition trees that well approximate the cuts in G. This set can be found using

the results by Räcke [21]. The reason why this process yields an $\mathcal{O}(\log n)$ approximation of the cut size depends on the properties of the decomposition, which we now detail, after introducing a few definitions. A *cut* $W \subseteq V$ of the vertices of a graph $G = (V, E, u)$, with capacity function $u : E \to \mathbb{R}^+$, is to be computed. The quality of the cut is measured using the *cut size* $C(W)$, which is the sum of the capacities of the edges connecting vertices in W and $V \setminus W$. A *decomposition tree* of a graph $G = (V, E, u)$ is a tree $T = (V_T, E_T, u_T)$, with capacity function $u_T : E_T \to \mathbb{R}^+$, for which the leaves $L \subset V_T$ of T correspond to the vertices in G. More precisely there is a mapping $m_G : V_T \to V$ of all tree vertices to vertices in G such that m_G induces a bijection between L and V. Let $m_T : V \to L$ denote the inverse function of this bijection. Accordingly we also define a *leaf cut* $K \subseteq L$ of a tree. The *cut size* $C(K)$ of a leaf cut K is the minimum capacity of edges in E_T that have to be removed in order to disconnect K from $L \setminus K$. We map cuts W to leaf cuts K and vice versa using the notation $m_T(W)$ and $m_G(K)$ to denote the image of W and K according to m_T and m_G respectively. We make use of the following result which can be found in [18, 21].

▶ **Theorem 9.** *For any graph $G = (V, E, u)$ with n vertices, a family of decomposition trees $\{T_i\}_i$ of G and positive real numbers $\{\lambda_i\}_i$ with $\sum_i \lambda_i = 1$ can be found in polynomial time, such that for any cut W of G and corresponding leaf cuts $K_i = m_{T_i}(W)$, $C(K_i) \geq C(W)$ for each i (lower bound), and $\sum_i \lambda_i C(K_i) \leq \mathcal{O}(\log n) \cdot C(W)$ (upper bound).*

Since $\sum_i \lambda_i = 1$ the above theorem implies that for at least one tree T_i it holds that $C(K_i) \leq \mathcal{O}(\log n) \cdot C(W)$. As we will see, this allows for a fast approximation of balanced partitions in graphs using a modified version of the PTAS given in Section 3. We adapt the PTAS to compute near-balanced *leaf partitions* of each T_i. That is, it computes a partition $\mathcal{L} = \{L_1, \ldots, L_k\}$ of the l leaves L of a tree T into k sets of size at most $(1 + \varepsilon)\lceil l/k \rceil$ each. The *cut size* $C(\mathcal{L})$ in this case is the minimum capacity of edges that have to be removed in order to disconnect the sets in $\mathcal{L}$ from each other. It is easy to see that the PTAS given in Section 3 can be adapted to compute near-balanced leaf partitions: first signatures need to count leaves instead of vertices in Definition 6. Also edge counts are replaced with sum of edge capacities, in Equation 2. Moreover, we need to keep track of the number l_v of leaves at a subtree of a vertex v instead of the number n_v of vertices in Equations 1 and 2.

▶ **Corollary 10.** *For any weighted tree T, $\varepsilon > 0$, and $k \in \{1, \ldots, l\}$, there is an algorithm that computes a partition of the l leaves of T into k sets such that each set includes at most $(1 + \varepsilon)\lceil l/k \rceil$ leaves and its cut size is at most that of an optimal perfectly balanced leaf partition. The runtime is polynomial in k and the number of vertices of T if ε is constant.*

Using the above results we show that near-balanced partitions that deviate by only a logarithmic factor from the optimal cut size can be computed for graphs in polynomial time.

▶ **Theorem 11.** *Let $G = (V, E, u)$ be a graph with n vertices, $\varepsilon > 0$ be a constant, and $k \in \{0, \ldots, n\}$. There is an algorithm that computes in polynomial time a partition of V into k sets such that each set has size at most $(1 + \varepsilon)\lceil n/k \rceil$ and its cut size is at most $\mathcal{O}(\log n)$ times that of the optimal perfectly balanced solution.*

5 Conclusions

In this paper, the k-`BALANCED PARTITIONING` problem was studied on trees and some of the results were applied to weighted graphs. When a perfectly balanced solution is required, we showed that even when either the diameter or the degree of the tree is constant the

k-BALANCED PARTITIONING problem remains hard to approximate. In this sense, trees represent the simplest unit that capture the full complexity of the problem.

On the other hand, if one settles for near-balanced solutions, trees prove to be "easy" instances which admit a PTAS with approximation $\alpha = 1$, the best possible in the bicriteria sense. This crucial fact enables our PTAS for trees to be extended into an algorithm for general graphs with approximation factor $\alpha \in \mathcal{O}(\log n)$, improving on the best previous [1] bound of $\alpha \in \mathcal{O}(\log^{1.5}(n)/\varepsilon^2)$. Hence, remarkably, the same approximation guarantee can be attained on the cut size for the k-BALANCED PARTITIONING problem in case $k = 2$ (the BISECTION problem) and for unrestricted k, if we settle for near-balanced solutions in the latter case. This is in contrast to the strong inapproximability results for both general graphs and trees when the solutions are to be perfectly balanced.

Open Problems. For perfectly balanced partitions of trees it remains open to generalise our results to show a tighter dependency of the hardness on the degree (Figure 5). In addition, the possibility of an approximation algorithm for perfectly balanced partitions with a better ratio than $\alpha \in \mathcal{O}(\Delta \log_\Delta(n/k))$, as provided by the greedy scheme by MacGregor [17], remains open. In particular, Theorem 3 does not rule out an algorithm that approximates the cut size by the factor $\alpha = 25/24$ if $\Delta \leq 7$.

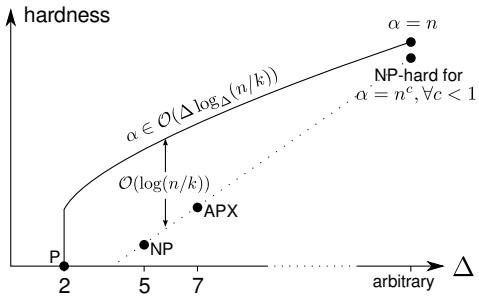


Figure 5 The hardness of trees versus their maximum degree Δ: the hardness results from Section 2 (large dots) indicate that the hardness grows with Δ (dotted line). A modification of MacGregor's [17] algorithm yields an approximation of $\mathcal{O}(\Delta \log_\Delta(n/k))$ (solid line). This means there is a gap of size $\mathcal{O}(\log(n/k))$ between the lower and upper complexity bounds in case Δ is constant.

For near-balanced partitions a question resulting from our work is whether faster algorithms can be found. We showed that we can achieve approximations on the cut size that do not depend on ε. Hence it suggests itself to try and find an algorithm that will compensate the cost of being able to compute a near-balanced solution for any $\varepsilon > 0$ not in the runtime but in the cut size. A recent result [6] however shows that there is no hope for a reasonable algorithm of this sort. More precisely it is shown that, unless P=NP, for trees there is no algorithm for which the runtime is polynomial in the input length and the inverse of ε, and has the following properties. The computed solution is near-balanced and the cut size may deviate from the optimum of a perfectly balanced solution by $\alpha = n^c/\varepsilon^d$, for any constants c and d where $c < 1$.

Another main challenge we see for general graphs is to resolve the discrepancy in complexity between the case $\varepsilon \geq 1$ and the case $\varepsilon < 1$, studied in this paper (recall Figure 2). For the case $\varepsilon \geq 1$ the algorithm by Krauthgamer *et al.* [15] achieves factor $\alpha \in \mathcal{O}(\sqrt{\log n \log k})$ and in the same paper it is shown that a dependency of α on k is unavoidable. Proving similar results for the case $\varepsilon < 1$ seems difficult to achieve, as the spreading metric techniques generally used for $\varepsilon \geq 1$ do not extend to $\varepsilon < 1$. Furthermore, the tree embedding results we used to achieve an $\mathcal{O}(\log n)$ approximation do not seem amenable to leading to algorithms with $o(\log n)$ approximation factor. Therefore it is likely that radically new techniques need to be developed to resolve the discrepancy.

References

1 K. Andreev and H. Räcke. Balanced graph partitioning. *Theor. Comp. Sys.*, 39(6):929–939, 2006.

2 S. Arora, S. Rao, and U. Vazirani. Expander flows, geometric embeddings and graph partitioning. In *Proceedings of the 26th annual ACM symposium on Theory of computing*, pages 222–231, 2004.

3 J. Díaz, J. Petit, and M. J. Serna. A survey of graph layout problems. *ACM Comput. Surv.*, 34(3):313–356, 2002.

4 J. Díaz, M. J. Serna, and J. Torán. Parallel approximation schemes for problems on planar graphs. *Acta Informatica*, 33(4):387–408, 1996.

5 G. Even, J. Naor, S. Rao, and B. Schieber. Fast approximate graph partitioning algorithms. In *Proceedings of the 8th annual ACM-SIAM symposium on Discrete algorithms*, pages 639–648, 1997.

6 A. E. Feldmann. Fast balanced partitioning of grid graphs is hard. *ArXiv e-prints*, (arXiv:1111.6745v1), November 2011.

7 A. E. Feldmann and L. Foschini. Balanced partitions of trees and applications. Technical Report 733, Institute of Theoretical Computer Science, ETH Zürich, July 2011.

8 A. E. Feldmann and P. Widmayer. An $O(n^4)$ time algorithm to compute the bisection width of solid grid graphs. In *Proceedings of the 19th Annual European Symposium on Algorithms*, pages 143–154, 2011.

9 T. Feo, O. Goldschmidt, and M. Khellaf. One-half approximation algorithms for the k-partition problem. *Operations Research*, 40:170–173, 1992.

10 T.A. Feo and M. Khellaf. A class of bounded approximation algorithms for graph partitioning. *Networks*, 20(2):181–195, 1990.

11 M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the Theory of NP-completeness*. W.H. Freeman and Co., 1979.

12 D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

13 D. G. Kirkpatrick and P. Hell. On the completeness of a generalized matching problem. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 240–245, 1978.

14 P. Klein, S.A. Plotkin, and S. Rao. Excluded minors, network decomposition, and multicommodity flow. In *Proceedings of the 25th annual ACM symposium on Theory of computing*, pages 682–690, 1993.

15 R. Krauthgamer, J. Naor, and R. Schwartz. Partitioning graphs into balanced components. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 942–949, 2009.

16 T. Leighton and S. Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.

17 R. M. MacGregor. *On partitioning a graph: a theoretical and empirical study*. PhD thesis, University of California, Berkeley, 1978.

18 A. Madry. Fast approximation algorithms for cut-based problems in undirected graphs. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 245 –254, 2010.

19 J. K. Park and C. A. Phillips. Finding minimum-quotient cuts in planar graphs. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 766–775, 1993.

20 E. Petrank. The hardness of approximation: Gap location. *Computational Complexity*, 4(2):133–157, 1994.

21 H. Räcke. Optimal hierarchical decompositions for congestion minimization in networks. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing*, 2008.

22 V. V. Vazirani. *Approximation Algorithms*. Springer, 2003.

Cache-Oblivious Implicit Predecessor Dictionaries with the Working-Set Property*

Gerth Stølting Brodal[1] and Casper Kejlberg-Rasmussen[1]

1 **MADALGO†, Department of Computer Science, Aarhus University, Denmark**

Abstract

In this paper we present an implicit dynamic dictionary with the working-set property, supporting insert(e) and delete(e) in $\mathcal{O}(\log n)$ time, predecessor(e) in $\mathcal{O}(\log \ell_{\mathsf{p}(e)})$ time, successor(e) in $\mathcal{O}(\log \ell_{\mathsf{s}(e)})$ time and search(e) in $\mathcal{O}(\log \min(\ell_{\mathsf{p}(e)}, \ell_e, \ell_{\mathsf{s}(e)}))$ time, where n is the number of elements stored in the dictionary, ℓ_e is the number of distinct elements searched for since element e was last searched for and $\mathsf{p}(e)$ and $\mathsf{s}(e)$ are the predecessor and successor of e, respectively. The time-bounds are all worst-case. The dictionary stores the elements in an array of size n using no additional space. In the cache-oblivious model the log is base B and the cache-obliviousness is due to our black box use of an existing cache-oblivious implicit dictionary. This is the first implicit dictionary supporting predecessor and successor searches in the working-set bound. Previous implicit structures required $\mathcal{O}(\log n)$ time.

1998 ACM Subject Classification Algorithms and data structures, E.1 Data Structures

Keywords and phrases working-set property, dictionary, implicit, cache-oblivious, worst-case, external memory, I/O efficient

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.112

1 Introduction

In this paper we consider the problem of maintaining a cache-oblivious implicit dictionary [11] with the working-set property over a dynamically changing set P of $|P| = n$ distinct and totally ordered elements. We define the *working-set number* of an element $e \in P$ to be $\ell_e = |\{e' \in P \mid$ we have searched for e' after we last searched for $e\}|$. An implicit dictionary maintains n distinct keys without using any other space than that of the n keys, i.e. the data structure is encoded by permuting the n elements. The fundamental trick in the implicit model, [10], is to encode a bit using two distinct elements x and y: if $\min(x, y)$ is before $\max(x, y)$ then x and y encode a 0 bit, else they encode a 1 bit. This can then be used to encode l bits using $2l$ elements. The implicit model is a restricted version of the unit cost RAM model with a word size of $\mathcal{O}(\log n)$. The restrictions are that between operations we are only allowed to use an array of the n input elements to store our data structures by permuting the input elements, i.e., there can be used no additional space between operations. In operations we are allowed to use $\mathcal{O}(1)$ extra words. Furthermore we assume that the number of elements n in the dictionary is externally maintained. Our structure will support the following operations:

- Search(e) determines if e is in the dictionary, if so its working-set number is set to 0.
- Predecessor(e) will find $\max\{e' \in P \cup \{-\infty\} \mid e' < e\}$, without changing any working-set numbers.

* This is an extended abstract, the full paper is available at `http://arxiv.org/abs/1112.5472`

† Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 112–123

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

Ref.	WS prop.	Insert/ Delete(e)	Search(e)	Pred(e)/ Succ(e)	Additional words
[10]	–	$\mathcal{O}(\log^2 n)$	$\mathcal{O}(\log^2 n)$	–	None
[5]	–	$\mathcal{O}\left(\frac{\log^2 n}{\log\log n}\right)$	$\mathcal{O}\left(\frac{\log^2 n}{\log\log n}\right)$	–	None
[7]	–	$\mathcal{O}(\log n)$ amor.	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	None
[6]	–	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$	None
[9]	+	$\mathcal{O}(\log n)$	$\mathcal{O}(\log \ell_e)$	$\mathcal{O}(\log \ell_{e^*})$	$\mathcal{O}(n)$
[3, Sec. 2]	+	$\mathcal{O}(\log n)$	$\mathcal{O}(\log \ell_e)$ exp.	$\mathcal{O}(\log n)$	$\mathcal{O}(\log\log n)$
[3, Sec. 3]	+	$\mathcal{O}(\log n)$	$\mathcal{O}(\log \ell_e)$ exp.	$\mathcal{O}(\log \ell_{e^*})$ exp.	$\mathcal{O}(\sqrt{n})$
[4]	+	$\mathcal{O}(\log n)$	$\mathcal{O}(\log \ell_e)$	$\mathcal{O}(\log n)$	None
This paper	+	$\mathcal{O}(\log n)$	$\mathcal{O}(\log \min(\ell_{\mathsf{p}(e)}, \ell_{\mathsf{s}(e)}, \ell_e))$	$\mathcal{O}(\log \ell_{e^*})$	None

Table 1 The operation time and space overhead of important structures for the dictionary problem. Here e^* is the predecessor or successor in the given context. In a search for an element e that is not present in the dictionary ℓ_e is n.

- Successor(e) will find $\min\{e' \in P \cup \{\infty\} \mid e < e'\}$, without changing any working-set numbers.
- Insert(e) inserts e into the dictionary with at working-set number of 0, all other working-set numbers are increased by one.
- Delete(e) deletes e from the dictionary, and does not change the working-set number of any element.

There has been a continuous development of implicit dictionaries, the first milestone was the implicit AVL-tree [10] having bounds of $\mathcal{O}(\log^2 n)$. The second milestone was the implicit B-tree [5] having bounds of $\mathcal{O}(\log^2 n/\log\log n)$ the third was the flat implicit tree [7] obtaining $\mathcal{O}(\log n)$ worst-case time for searching and amortized bounds for updates. The fourth milestone is the optimal implicit dictionary [6] obtaining worst-case $\mathcal{O}(\log n)$ for search, update, predecessor and successor.

Numerous non-implicit dictionaries attain the working-set property; splay trees [12], skip list variants [2], the working-set structure in [9], and two structures presented in [3]. All achieve the property in the amortized, expected or worst-case sense. The unified access bound, which is achieved in [1], even combines the working-set property with finger search. In finger search we have a finger located on an element f and the search cost of finding say element e is a function of $d(f, e)$ which is the rank distance between elements f and e. The unified bound combines these two to obtain a bound of $\mathcal{O}(\min_{e\in P}\{\log(\ell_e + d(e, f) + 2)\})$. Table 1 gives an overview of previous results, and our contribution.

The dictionary in [6] is, in addition to being implicit, also designed for the cache-oblivious model [8], where all the operations imply $\mathcal{O}(\log_B n)$ cache-misses. Here B is the cache-line length which is unknown to the algorithm. The cache-oblivious property also carries over into our dictionary. Our structure combines the two worlds of implicit dictionaries and dictionaries with the working-set property to obtain the first implicit dictionary with the working-set property supporting search, predecessor and successor queries in the working-set bound. The result of this paper is summarized in Theorem 1.

▶ Theorem 1. There exists a cache-oblivious implicit dynamic dictionary with the working-set property that supports the operations insert and delete in time $\mathcal{O}(\log n)$ and $\mathcal{O}(\log_B n)$ cache-misses, search, predecessor and successor in time $\mathcal{O}(\log \min(\ell_{\mathsf{p}(e)}, \ell_e, \ell_{\mathsf{s}(e)}))$, $\mathcal{O}(\log \ell_{\mathsf{p}(e)})$ and $\mathcal{O}(\log \ell_{\mathsf{s}(e)})$, and cache-misses $\mathcal{O}(\log_B \min(\ell_{\mathsf{p}(e)}, \ell_e, \ell_{\mathsf{s}(e)}))$, $\mathcal{O}(\log_B \ell_{\mathsf{p}(e)})$ and $\mathcal{O}(\log_B \ell_{\mathsf{s}(e)})$,

respectively, where $\mathsf{p}(e)$ and $\mathsf{s}(e)$ are the predecessor and successor of e, respectively.

Similarly to previous work [1, 4] we partition the dictionary elements into $\mathcal{O}(\log\log n)$ blocks $B_0, \ldots, B_m$, of double exponential increasing sizes, where B_0 stores the most recently accessed elements. The structure in [4] supports predecessors and successors queries, but there is no way of knowing if an element is actually the predecessor or successor, without querying all blocks, which results in $\mathcal{O}(\log n)$ time bounds. We solve this problem by introducing the notion of *intervals* and particularly a dynamic implicit representation of these. We represent the whole interval $[\min(P); \max(P)]$ by a set of disjoint intervals spread across the different blocks. Any point that intersects an interval in block B_i will lie in block B_i and have a working-set number of at least 2^{2^i}. This way when we search for the predecessor or successor of an element and hit an interval, then no more points can be contained in the interval in higher blocks, and we can avoid looking at these, which give working-set bounds for the search, predecessor and successor queries.

2 Data structure

We now describe our data structure and its invariants. We will use the moveable dictionary from [4] as a black box. The dictionary over a point set S is laid out in the memory addresses $[i; j]$. It supports the following operations in $\mathcal{O}(\log n')$ time and $\mathcal{O}(\log_B n')$ cache-misses, where $n' = j - i + 1$:

- Insert-left(e) inserts e into S which is now laid out in the addresses $[i-1; j]$.
- Insert-right(e) inserts e into S which is now laid out in the addresses $[i; j+1]$.
- Delete-left(e) deletes e from S which is now laid out in the addresses $[i+1; j]$.
- Delete-right(e) deletes e from S which is now laid out in the addresses $[i; j-1]$.
- Search(e) determines if $e \in S$, if so the address of element e is returned.
- Predecessor(e) returns the address of the element $\max\{e' \in S \mid e' < e\}$ or that no such element exists.
- Successor(e) returns the address of the element $\min\{e' \in S \mid e < e'\}$ or that no such element exists.

From these operations we notice that we can move the moveable dictionary, say left, by performing a delete-right operation for an arbitrary element and re-inserting the element again by an insert-left operation. Similarly we can also move the dictionary one position to the right.

Our structure consists of $m = \Theta(\log\log n)$ blocks $B_0, \ldots, B_m$, each block B_i is of size $\mathcal{O}(2^{2^{i+k}})$, where k is a constant. Elements in B_i have a working-set number of at least $2^{2^{i+k-1}}$. The block B_i consists of an array D_i of $w_i = d \cdot 2^{i+k}$ elements, where d is a constant, and moveable dictionaries A_i, R_i, W_i, H_i, C_i and G_i, for $i = 0, \ldots, m-1$, see Figure 1. For block B_m we only have D_m if $|B_m \backslash \{\min(P), \max(P)\}| \leq w_m$, otherwise we have the same structures as for the other blocks. We use the block D_i to encode the sizes of the movable dictionaries A_i, R_i, W_i, H_i, C_i and G_i so that we can locate them. Discussion of further details of the memory layout is postponed to Section 3.

We call elements in the structures D_i and A_i for *arriving* points, and when making a non-arriving point arriving, we will put it into A_i unless specified otherwise. We call elements in R_i for *resting* points, elements in W_i for *waiting* points, elements in H_i for *helping* points, elements in C_i for *climbing* points and elements in G_i for *guarding* points.

Crucial to our data structure is the partitioning of $[\min(P); \max(P)]$ into *intervals*. Each interval is assigned to a *level* and level i corresponds to block B_i. Consider an interval lying at level i. The endpoints e_1 and e_2 will be guarding points stored at level $0, \ldots, i$.

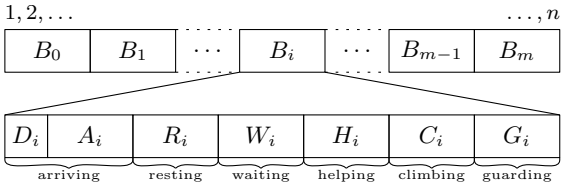


Figure 1 Overview of how the working set dictionary is laid out in memory. The dictionary grows and shrinks to the right when elements are inserted and deleted.

All points inside of this interval will lie in level i and cannot be guarding points, i.e. $]e_1; e_2[\cap(\bigcup_{j\neq i} B_j \cup G_i) = \emptyset$. We do not allow intervals defined by two consecutive guarding points to be empty, they must contain at least one non-guarding point. We also require $\min(P)$ and $\max(P)$ to be guarding points in G_0 at level 0, but they are special as they do not define intervals to their left and right, respectively. A query considers $B_0, B_1, \ldots$ until B_i where the query is found to be in a level i interval where the answer is guaranteed to have been found in blocks $B_0, \ldots, B_i$.

The basic idea of our construction is the following. When searching for an element it is moved to level 0. This can cause block overflows (see invariants I.5–I.9 in Section 2.2), which are handled as follows. The arriving points in level i have just entered from level $i-1$, and when there are $2^{2^{i+k}}$ of them in A_i they become resting. The resting points need to charge up their working-set number before they can begin their journey to level $i+1$. They are charged up when there have come $2^{2^{i+k}}$ further arriving points to level i, then the resting points become waiting points. Waiting points have a high enough working-set number to begin the journey to level $i+1$, but they need to wait for enough points to group up so that they can start the journey. When a waiting point is picked to start its journey to level $i+1$ it becomes a helping or climbing point, and every time enough helping points have grouped up, i.e. there is at least $c = 5$ consecutive of them, then they become climbing points and are ready to go to level $i+1$. The climbing points will then incrementally be going to level $i+1$.

2.1 Notation

Before we introduce the invariants we need to define some notation. For a subset $S \subseteq P$, we define $\mathsf{p}_S(e) = \max\{s \in S \cup \{-\infty\} \mid s < e\}$ and $\mathsf{s}_S(e) = \min\{s \in S \cup \{\infty\} \mid e < s\}$. When we write $S_{\leq i}$ we mean $\bigcup_{j=0}^{i} S_j$ where $S_j \subseteq P$ for $j = 0, \ldots, i$.

For $S \subseteq P$, we define $\mathsf{GIL}_S(e) = S\cap]\mathsf{p}_{P\setminus S}(e); e[$ to be the Group of Immediate Left points of e in S which does not have any other point of $P\setminus S$ in between them. Similarly we define $\mathsf{GIR}_S(e) = S\cap]e; \mathsf{s}_{P\setminus S}(e)[$ to the right of e. We will notice that we will never find all points of $\mathsf{GIL}_S(e)$ unless $|\mathsf{GIL}_S(e)| < c$, the same applies for $\mathsf{GIR}_S(e)$. For $S \subseteq P$, we define $\mathsf{FGL}_S(e) = S\cap]\mathsf{p}_{P\setminus S}(\mathsf{p}_S(e)); \mathsf{p}_S(e)]$ to be the First Group of points from S Left of e, i.e. the group does not have any points of $P\setminus S$ in between its points. Similarly we define $\mathsf{FGR}_S(e) = S \cap [\mathsf{s}_S(e); \mathsf{s}_{P\setminus S}(\mathsf{s}_S(e))[$. We will notice that we will never find all points of $\mathsf{FGL}_S(e)$ unless $|\mathsf{FGL}_S(e)| < c$, the same applies for $\mathsf{FGR}_S(e)$.

We will sometimes use the phrasings *a group of points* or *e's group of points*. This refers to a group of points of the same type, i.e. arriving, resting, etc., and with no other types of points in between them. Later we will need to move elements around between the structures D_i, A_i, R_i, W_i, H_i, C_i and G_i. For this we have the notation $X \xrightarrow{h} Y$, meaning that we move h arbitrary points from X into Y, where X and Y can be one of D_i, A_i, R_i, W_i, H_i, C_i and G_i for any i.

When we describe the intervals we let $]a;b]$ be an interval from a to b that is open at a and closed at b. We let $(a;b)$ be an interval from a to b that can be open or closed at a and b. We use this notation when we do not care if a and b are open or closed. In the methods updating the intervals we will sometimes branch depending on which type an interval is. For clarity we will explain how to determine this given the level i of the interval and its two endpoints e_1 and e_2. The interval $(e_1;e_2)$ is of type $[e_1;e_2)$ if $e_1 \in G_i$, else $e_1 \in G_{\leq i-1}$ and the interval is of type $]e_1;e_2)$. This is symmetric for the other endpoint e_2.

2.2 Invariants

We will now define the invariants which will help us define and prove correctness of our interface operations: insert(e), delete(e), search(e), predecessor(e) and successor(e). We maintain the following invariants which uniquely determine the intervals[1]:

I.1 A guarding point is part of the definition of at most two intervals[2], one to the left at level i and/or one to the right at level j, where $i \neq j$. The guarding point e lies at level $\min(i,j)$. The interval at level $\min(i,j)$ is closed at e, and the interval at level $\max(i,j)$ is open at e. We also require that $\min(P)$ and $\max(P)$ are guarding points stored in G_0, but they do not define an interval to their left and right, respectively, and the intervals they help define are open in the end they define. A non-guarding point intersecting an interval at level i, lies in level i. Each interval contains at least one non-guarding point. The union of all intervals give $]\min(P);\max(P)[$.

I.2 Any climbing point, which lies in an interval with other non-climbing points, is part of a group of at least c points. In intervals of type $[e_1;e_2]$ which only contain climbing points, we allow there to be less than c of them.

I.3 Any helping point is part of a group of size at most $c-1$. A helping point cannot have a climbing point as a predecessor or successor. An interval of type $[e_1;e_2]$ cannot contain only helping points.

We maintain the following invariants for the working-set numbers:

I.4 Each arriving point in D_i and A_i has a working set value of at least $2^{2^{i-1+k}}$, arriving points in D_0 and A_0 have a working-set value of at least 0. Each resting point in R_i will have a working-set value of at least $2^{2^{i-1+k}} + |A_i|$, resting points in R_0 have a working-set value of at least $|A_0|$. Each waiting, helping or climbing point in W_i, H_i and C_i, respectively, will have a working-set value of at least $2^{2^{i+k}}$. Each guarding point in G_i, who's left interval lies at level i and right interval lies at level j, has a working set value of at least $2^{2^{\max(i,j)-1+k}}$.

We maintain the following invariants for the size of each block and their components:

I.5 $|D_0| = \min(|B_0|-2, w_0)$ and $|D_i| = \min(|B_i|, w_i)$ for $i = 1,\ldots,m$.
I.6 $|R_i| \leq 2^{2^{i+k}}$ and $|W_i| + |H_i| + |C_i| \neq 0 \Rightarrow |R_i| = 2^{2^{i+k}}$ for $i = 0,\ldots,m$.
I.7 $|A_i| + |W_i| = 2^{2^{i+k}}$ for $i = 0,\ldots,m-1$, and $|A_m| + |W_m| \leq 2^{2^{m+k}}$.
I.8 $|A_i| < 2^{2^{i+k}}$ for $i = 0,\ldots,m$.
I.9 $|H_i| + |C_i| = 4c2^{2^{i+k}} + c_i$, where $c_i \in [-c;c]$, for $i = 0,\ldots,m-1$.

---

[1] We assume that $|P| = n \geq 2$ at all times if this is not the case we only store G_0 which contains a single element and we ignore all invariants.

[2] Only the smallest and largest guarding points will not participate in the definition of two intervals, all other guarding points will.

From the above invariants we have the following observation:

O.1 From I.1 all points in G_i are endpoints of intervals in level i, and each interval has at most two endpoints. Hence for $i = 0, \ldots, m$ we have that

$$|G_i| \leq 2(|D_i| + |A_i| + |R_i| + |W_i| + |H_i| + |C_i|) \overset{(*)}{\leq} (4 + 2d + 8c) \cdot 2^{2^{i+k}} + 2c \,,$$

where we in $(*)$ we have used I.5, I.6, I.7 and I.9.

From I.1 we have the following lemma.

▶ Lemma 1. Let e be an element, $e_1 = \mathsf{p}_{G_{\leq i}}(e)$, $e_2 = \mathsf{s}_{G_{\leq i}}(e)$ and i be the smallest integer for which $I(e_1, e_2, i) =]e_1; e_2[\cap \bigcup_{j=0}^{i} B_j \neq \emptyset$. Then 1) $(e_1; e_2)$ is an interval at level i if e is non-guarding and 2) $(e_1; e)$ or $(e; e_2)$ is an interval at level i if e is guarding.

2.3 Operations

We will briefly give an overview of the helper operations and state their requirements *(R)* and guarantees *(G)*, then we will describe the helper and interface operations in details. Search(e) uses the helper operations as follows: when a search for element e is performed then the level i where e lies is found using find, then e and $\mathcal{O}(1)$ of its surrounding elements are moved into level 0 by use of move-down while maintaining I.1–I.4. Calls to fix for the levels we have altered will ensure that I.5–I.8 will be maintained, finally a call to rebalance-below$(i-1)$ will ensure that I.9 is maintained by use of shift-up(j) which will take climbing points from level j and make them arriving in level $j+1$ for $j = 0, \ldots, i-1$. Insert(e) uses find to find the level where e intersects, then it uses fix to ensure the size constraints and finally e is moved to level 0 by use of search.

- Find(e) - returns the level i of the interval that e intersects along with e's type and whatever e is in the dictionary or not. *[R&G: I.1–I.9]*
- Fix(i) - moves points around inside of B_i to ensure the size invariants for each type of point. Fix(i) might violate I.9 for level i. *[R: I.1–I.4 and that there exist $\tilde{c}_1, \ldots, \tilde{c}_6$ such that $|D_i| + \tilde{c}_1, |A_i| + \tilde{c}_2, |R_i| + \tilde{c}_3, |W_i| + \tilde{c}_4, |H_i| + \tilde{c}_5, |C_i| + \tilde{c}_6$ fulfill I.5–I.8, where $|\tilde{c}_i| = \mathcal{O}(1)$ for $i = 1, \ldots, 6$. G: I.1–I.8].*
- Shift-down(i) - will move at least 1 and at most c points from level i into level $i-1$. *[R: I.1–I.8 and $|H_i| + |C_i| = 4c2^{2^{i+k}} + c'_i$, where $0 \leq c'_i = \mathcal{O}(1)$. G: I.1–I.8].*
- Shift-up(i) - will move at least 1 and at most c points from level i into level $i+1$. *[R: I.1–I.8 and $|H_i| + |C_i| = 4c2^{2^{i+k}} + c'_i$, where $c \leq c'_i = \mathcal{O}(1)$. G: I.1–I.8].*
- Move-down$(e, i, j, t_{\text{before}}, t_{\text{after}})$ - If e is in the dictionary at level i it is moved from level i to level j, where $i \geq j$. The type t_{before} is the type of e before the move and t_{after} is the type that e should have after the move, unless $i = j$ in which case e will be made arriving in level j. *[R&G: I.1–I.8].*
- Rebalance-below(i) - If any $c < c_l$ for $l = 0, \ldots, i$ rebalance-below(i) will correct it so I.9 will be fulfilled again for $l = 0, \ldots, i$. *[R: I.1–I.8 and $\sum_{l=0}^{i} \mathit{slack}(c_l) = \mathcal{O}(1)$, where*

$$\mathit{slack}(c_l) = \begin{cases} 0 & \text{if} \quad c_l \in [-c; c] \,, \\ |c_l| - c & \text{otherwise} \,. \end{cases}$$

G: I.1–I.9].
- Rebalance-above(i) - If any $c_l < -c$ for $l = i, \ldots, m-1$ rebalance-above(i) will correct it so I.9 will be fulfilled again for $l = i, \ldots, m-1$. *[R: I.1–I.8 and $\sum_{l=i}^{m-1} \mathit{slack}(c_l) = \mathcal{O}(1)$. G: I.1–I.9].*

Find(e) We start at level $i = 0$. If $e < \min(P)$ or $\max(P) < e$ we return false and 0. For each level we let $e_1 = \mathsf{p}_{G_{\leq i}}(e)$, $e_2 = \mathsf{s}_{G_{\leq i}}(e)$, $p = \mathsf{p}_{B_i \setminus G_i}(e)$ and $s = \mathsf{s}_{B_i \setminus G_i}(e)$. We find p and s by querying each of the structures D_i, A_i, R_i, W_i, H_i and C_i, we find e_1 and e_2 by querying G_i and comparing with the values of e_1 and e_2 from level $i-1$. While $p < e_1$ and $e_2 < s$ we continue to the next level, that is we increment i. Now outside the loop, if $e \in B_i$ we return i, the type of e and the boolean true as we found e, else we return i and false as we did not find e.

Predecessor(e) **(successor**(e)**)** We start at level $i = 0$. If $e < \min(P)$ then return $-\infty$ $(\min(P))$. If $\max(P) < e$ then return $\max(P)$ (∞). For each level we let $e_1 = \mathsf{p}_{G_{\leq i}}(e)$, $p = \mathsf{p}_{B_i}(e)$, $e_2 = \mathsf{s}_{G_{\leq i}}(e)$ and $s = \mathsf{s}_{B_i}(e)$. While $p < e_1$ and $e_2 < s$ we continue to the next level, that is we increment i. When the loop breaks we return $\max(e_1, p)$ $(\min(s, e_2))$.

Insert(e) If $e < \min(P)$ we swap e and $\min(P)$, call fix(0), rebalance-below(m) and return. If $\max(P) < e$ we swap e and $\max(P)$, call fix(0), rebalance-below(m) and return.

Let $c_l = \mathsf{GIL}_{C_i}(e)$, $c_r = \mathsf{GIR}_{C_i}(e)$, $h_l = \mathsf{GIL}_{H_i}(e)$ and $h_r = \mathsf{GIR}_{H_i}(e)$. We find the level i of the interval $(e_1; e_2)$ which e intersects using find(e).

If e is already in the dictionary we give an error. If $|c_l| > 0$ or $|c_r| > 0$ or $(e_1; e_2)$ is of type $[e_1; e_2]$ and does not contain non-climbing points then insert e as climbing at level i. Else if $|h_l| + 1 + |h_r| \geq c$ then insert e as climbing at level i and make the points in h_l and h_r climbing at level i. Else insert e as helping at level i. Finally we call rebalance-below(m) and then search(e) to move e from the current level i down to level 0.

Search(e) We first find e's current level i and its type t, by a call to find(e). If e is in the dictionary then we call move-down$(e, i, 0, t,$ arriving$)$ which will move e from level i down to level 0 and make it arriving, while maintaining I.1–I.8, but I.9 might be broken so we finally call rebalance-below$(i-1)$ to fix this.

Fix(i) In the following we will be moving elements around between D_i, A_i, R_i, W_i, H_i and C_i. The moves $A_i \to R_i$ and $R_i \to W_i$, i.e. between structures which are next to each other in the memory layout, are simply performed by deleting an element from the left structure and inserting it into the right structure. The moves $W_i \to H_i \cup C_i$ and the other way around $H_i \cup C_i \to W_i$ will be explained below.

If $|D_i| > w_i$ then perform $D_i \xrightarrow{h} A_i$ where $h = |D_i| - w_i$. If $|D_i| < w_i$ and $|B_i \setminus \{\min(P), \max(P)\}| > |D_i|$ then perform $H_i \cup C_i \xrightarrow{h_1} W_i$, $W_i \xrightarrow{h_2} R_i$, $R_i \xrightarrow{h_3} A_i$ and $A_i \xrightarrow{h_4} D_i$ where $h_1 = \min(w_i - |D_i|, |H_i| + |C_i|)$, $h_2 = \min(w_i - |D_i|, |W_i| + h_1)$, $h_3 = \min(w_i - |D_i|, |R_i| + h_2)$ and $h_4 = \min(w_i - |D_i|, |A_i| + h_3)$.

If $|W_i| + |H_i| + |C_i| \neq 0$ and $|R_i| < 2^{2^{i+k}}$ then perform $H_i \cup C_i \xrightarrow{h_1} W_i$ and $W_i \xrightarrow{h_2} R_i$ where $h_1 = \min(2^{2^{i+k}} - |R_i|, |H_i| + |C_i|)$ and $h_2 = \min(2^{2^{i+k}} - |R_i|, |W_i| + h_1)$. If $|R_i| > 2^{2^{i+k}}$ then perform $R_i \xrightarrow{h_1} A_i$ where $h_1 = |R_i| - 2^{2^{i+k}}$.

If $i < m$ and $|A_i| + |W_i| < 2^{2^{i+k}}$ then perform $H_i \cup C_i \xrightarrow{h_1} W_i$, where $h_1 = \min(2^{2^{i+k}} - (|A_i| + |W_i|), |H_i| + |C_i|)$. If $|A_i| + |W_i| > 2^{2^{i+k}}$ then perform $W_i \xrightarrow{h_1} H_i \cup C_i$ where $h_1 = \min(|A_i| + |W_i| - 2^{2^{i+k}}, |W_i|)$.

If $|A_i| \geq 2^{2^{i+k}}$ then let $h_1 = |A_i| - 2^{2^{i+k}}$, delete W_i as it is empty and rename R_i to W_i. Now move h_1 elements from A_i into a new moveable dictionary X, rename A_i to R_i, rename X to A_i and perform $W_i \xrightarrow{h_1} H_i \cup C_i$.

Performing $W_i \to H_i \cup C_i$*:* Let $w = \mathsf{s}_{W_i}(-\infty)$, $c_l = \mathsf{GIL}_{C_i}(w)$, $c_r = \mathsf{GIR}_{C_i}(w)$, $h_l = \mathsf{GIL}_{H_i}(w)$ and $h_r = \mathsf{GIR}_{H_i}(w)$. If $|c_l| > 0$ or $|c_r| > 0$ or $(e_1; e_2)$ is of type $[e_1; e_2]$ and only contains climbing points then make w climbing at level i. Else if $|h_l| + 1 + |h_r| \geq c$ then make h_l, w and h_r climbing at level i. Else make w helping at level i.

Performing $H_i \cup C_i \to W_i$*:* Let w be the minimum element of $\mathsf{s}_{H_i}(-\infty)$ and $\mathsf{s}_{C_i}(-\infty)$, and let $c_r = \mathsf{GIR}_{C_i}(w)$. Make w waiting at level i. If w was climbing and $|c_r| < c$ then make c_r helping at level i.

Shift-down(i) We move at least one element from level i into level $i-1$. If $|D_i| < w_i$ then we let a be some element in D_i. If $|D_i| < |B_i|$ then: if $|A_i| = 0$ we perform[3] $H_i \cup C_i \xrightarrow{h_1} W_i$, $W_i \xrightarrow{h_2} R_i$ and $R_i \to A_i$, where $h_1 = \min(1, |H_i| + |C_i|)$ and $h_2 = \min(1, |W_i| + h_1)$, now we know that $|A_i| > 0$ so let $a = \mathsf{s}_{A_i}(-\infty)$, i.e., a is the leftmost arriving point in A_i at level i. We call move-down$(a, i, i-1, \text{arriving}, \text{climbing})$.

Shift-up(i) Assume we are at level i, we want to move at least one and at most c arbitrary points from B_i into B_{i+1}. Let $s_1 = \mathsf{s}_{C_i}(-\infty)$, $e_1 = \mathsf{p}_{G_{\leq i}}(s_1)$ and $e_2 = \mathsf{s}_{G_{\leq i}}(s_1)$, and let $s_2 = \mathsf{s}_{C_i \cap [e_1; e_2]}(s_1)$, $s_3 = \mathsf{s}_{C_i \cap [e_1; e_2]}(s_2)$, $s_4 = \mathsf{s}_{C_i \cap [e_1; e_2]}(s_3)$ and $s_5 = \mathsf{s}_{C_i \cap [e_1; e_2]}(s_4)$, if they exist, also let $c_r = \mathsf{GIR}_{C_i}(s_4)$ be the group of climbing elements to the immediate right of s_4, if they exist. We will now move one or more climbing points from B_i into B_{i+1} where they become arriving points. If $i = m - 1$ or $i = m$ then we put arriving points into D_{i+1}, which we might have to create, instead of A_{i+1}.

We now deal with the case where $(e_1; e_2)$ is of type $[e_1; e_2]$ and only contains climbing points. Let l be the level of e_1's left interval, and r the level of e_2's right interval, also let c_I be the number of climbing points in the interval. If $l = i + 1$ we make e_1 arriving, else we make it guarding, at level $i + 1$. Make the points of s_1, s_2, s_3 and s_4 that exist arriving at level $i + 1$. If $c_I \leq c$ then make s_5 arriving at level $i + 1$ if it exists, also if $r = i + 1$ we make e_2 arriving, else we make it guarding, at level $i + 1$. Else make s_5 guarding at level i.

We now deal with the cases where $(e_1; e_2)$ might contain non-climbing points. If $\mathsf{p}(s_1) = e_1$ we make s_1 and s_2 waiting and guarding at level i, respectively, else we make s_1 guarding at level i and s_2 arriving at level $i + 1$. Now in both cases we make s_3 arriving at level $i + 1$ and s_4 guarding at level i. If $\langle (s_4; e_2)$ is not of type $[s_4; e_2]$ or contains non-climbing points$\rangle$ and $|c_r| < c$, i.e. there are less than c consecutive climbing points to the right of s_4, then we make the points c_r helping at level i.

We have moved climbing points from B_i into B_{i+1}, and made them arriving. Finally we call fix$(i + 1)$.

Move-down$(e, i, j, t_{\textbf{before}}, t_{\textbf{after}})$ Depending on the type t_{before} of point e we have different cases.

Non-guarding Let $e_1 = \mathsf{p}_{G_{\leq i}}(e)$, $e_2 = \mathsf{s}_{G_{\leq i}}(e)$ and let l be the level of the left interval of e_1 and r the level of the right interval of e_2. Also let $p_2 = \mathsf{p}_{B_i \setminus G_i \cap [e_1; e_2]}(p_1)$, $p_1 = \mathsf{p}_{B_i \setminus G_i \cap [e_1; e_2]}(e)$, $s_1 = \mathsf{s}_{B_i \setminus G_i \cap [e_1; e_2]}(e)$ and $s_2 = \mathsf{s}_{B_i \setminus G_i \cap [e_1; e_2]}(s_1)$, also let $c_l = \mathsf{FGL}_{C_i \cap [e_1; e_2]}(e)$ be the elements in the first climbing group left of e, likewise let $c_r = \mathsf{FGR}_{C_i \cap [e_1; e_2]}(e)$ be the elements in the first climbing group right of e.

[3] The move $H_i \cup C_i \xrightarrow{l} W_i$ will be performed the same way as we did it in fix.

Case $i = j$: make e arriving in level j, if $|c_l| < c$ then make the points in c_l helping at level j, if $|c_r| < c$ then make the points in c_r helping at level j. Finally call fix(j).

Case $i > j$: If both p_2 and p_1 exists we make p_1 guarding in level j and let e'_1 denote p_1, else if only p_1 exists we make e_1 guarding at level $\min(l, j)$ and p_1 of type t_{after} at level j and let e'_1 denote e_1, else we make e_1 guarding in level $\min(l, j)$, and let e'_1 denote e_1. If both s_1 and s_2 exists we make s_1 guarding at level j, and let e'_2 denote s_1, else if only s_1 exists we make s_1 of type t_{after} at level j and make e_2 guarding at level $\min(j, r)$ and let e'_2 denote e_2, else we make e_2 guarding at level $\min(j, r)$ and let e'_2 denote e_2. Lastly we make e of type t_{after} in level j. Now let c'_l denote the elements of c_l which we have not moved in the previous steps, likewise let c'_r denote the elements of c_r which we have not moved. If $\langle(e_1; e'_1]$ is not of type $[e_1; e'_1]$ or contains non-climbing points$\rangle$ and $|c'_l| < c$ then make c'_l helping at level i. If $\langle[e'_2; e_2)$ is not of type $[e'_2; e_2]$ or contains non-climbing points$\rangle$ and $|c'_r| < c$ then make c'_r helping at level i. Call fix(i), fix(j), fix($\min(l, i)$) and fix($\min(i, r)$).

Guarding If $e = \min(P)$ or $e = \max(P)$ we simply do nothing and return. Let $e_1 = \mathsf{p}_{G_{\leq h}}(e)$ be the left endpoint of the left interval $(e_1; e[$ lying at level h and $e_2 = \mathsf{s}_{G_{\leq h}}(e)$ be the right endpoint of the right interval $[e; e_2)$ lying at level i, we assume w.l.o.g. that $h > i$, the case $h < i$ is symmetric. Also let l be the level of the left interval of e_1 and r the level of the right interval of e_2. Let $p_2 = \mathsf{p}_{B_h \setminus G_h \cap [e_1; e]}(p_1)$ and $p_1 = \mathsf{p}_{B_h \setminus G_h \cap [e_1; e]}(e)$ be the two left points of e, if they exists, $s_1 = \mathsf{s}_{B_i \setminus G_i \cap [e; e_2]}(e)$ and $s_2 = \mathsf{s}_{B_i \setminus G_i \cap [e; e_2]}(s_1)$ the two right points of e, if they exits. Also let $c_l = \mathsf{FGL}_{C_i \cap [e_1; e]}(e)$ and $c_r = \mathsf{FGR}_{C_i \cap [e; e_2]}(e)$.

If p_2 does not exist we make e_1 guarding at level $\min(l, j)$, we make p_1 of type t_{after} at level j and let e'_1 denote e_1, else we make p_1 guarding at level j and let e'_1 denote p_1. If it is the case that $i > j$ then we check: if s_2 does not exist then we make s_1 of type t_{after} at level j, e_2 guarding at level $\min(j, r)$ and let e'_2 denote e_2, else we make s_1 guarding at level j and let e'_2 denote s_1. We make e of type t_{after} at level j.

Now let c'_l be the points of c_l which was not moved and c'_r the points of c_r which was not moved. If $|c'_l| < c$ then make c'_l helping at level h. We now have two cases if e'_2 exists: then if $|c'_r| < c$ then make c'_r helping at level i. The other case is if e'_2 does not exist: then if $\langle(e'_1; e_2)$ is not of type $[e'_1; e_2]$ or contains non-climbing points$\rangle$ and $|c'_r| < c$ then make c'_r helping at level i. In all cases call fix($\min(l, h)$), fix(h) and fix(i). If $i > j$ then call fix(j) and fix($\min(j, r)$).

Delete(e) We first call find(e) to get the type of e and its level i, if e is not in the dictionary we just return. If e is in the dictionary we have two cases, depending on if e is guarding or not.

Non-guarding Let $c_l = \mathsf{GIL}_{C_i}(e)$ be the elements in the climbing group immediately left of e, let $c_r = \mathsf{GIR}_{C_i}(e)$ be the elements in the climbing group immediately right of e, let $h_l = \mathsf{GIL}_{H_i}(e)$ be the elements in the helping group immediately left of e, and let $h_r = \mathsf{GIR}_{H_i}(e)$ be the elements in the helping group immediately right of e. Let $e_1 = \mathsf{p}_{G_{\leq i}}(e)$ and let $e_2 = \mathsf{s}_{G_{\leq i}}(e)$. Let l be the level of the interval left of e_1 and r the level of the interval right of e_2.

We have two cases, the first is $|]e_1; e_2[\cap B_i| = 1$: if $l > r$ make e_1 guarding and e_2 arriving at level r, if $l < r$ then make e_1 arriving and e_2 guarding at level l. If $l = r$ and $|P| = n \geq 4$ then make e_1 and e_2 arriving at level $l = r$. Delete e, call fix(r), fix(l), fix(i) and rebalance-above(1).

The other case is $|]e_1; e_2[\cap B_i| > 1$: If $\langle(e_1; e_2)$ is not of type $[e_1; e_2]$ or contains non-climbing points$\rangle$ and $|c_l| + |c_r| < c$ then make c_l and c_r helping at level i. If $|h_l| + |h_r| \geq c$ then make h_l and h_r climbing at level i. Delete e, call fix(i) and rebalance-above(1).

Min-guarding If $e = \min(P)$ then let $e' = \mathsf{s}_{G_{\leq m}}(e)$ and $e'' = \mathsf{s}_{G_{\leq m}}(e')$ where 0 is the level of $(e; e')$ and i is the level of $(e'; e'')$. The case of $e = \max(P)$ is symmetric. Also let $s_1 = \mathsf{s}_{B_0 \setminus G_0 \cap [e;e']}(e)$, $s_2 = \mathsf{s}_{B_0 \setminus G_0 \cap [e;e']}(s_1)$, $t_1 = \mathsf{s}_{B_i \setminus G_i \cap [e';e'']}(e')$ and $t_2 = \mathsf{s}_{B_i \setminus G_i \cap [e';e'']}(t_1)$.

If s_2 exists then delete e make s_1 guarding at level 0 and call fix(0). If s_2 does not exist and t_2 exists then delete e make s_1 and t_1 guarding and e' arriving at level 0 and finally call fix(0) and fix(i). If s_2 does not exist and t_2 does not exist then delete e, make s_1 and e'' guarding and e' and t_1 arriving at level 0 and finally call fix(0) and fix(i). In all the previous cases return.

Guarding Let h be the level of the left interval $(e_1 : e[$, let i the level of the right interval $[e : e_2)$ that e participates in. We assume w.l.o.g. that $h > i$, the case $h < i$ is symmetric. Let l the level of the left interval that e_1 participates in, where $e_1 = \mathsf{p}_{G_{\leq h}}(e)$ and $e_2 = \mathsf{s}_{G_{\leq h}}(e)$. Let $p_2 = \mathsf{p}_{B_h \setminus G_h \cap [e_1;e]}(p_1)$ and $p_1 = \mathsf{p}_{B_h \setminus G_h \cap [e_1;e]}(e)$. Let $c_l = \mathsf{FGL}_{C_i}(e)$ be the points in the first group of climbing points left of e.

If p_2 exist we make p_1 guarding at level i, and let e' denote p_1, else we make e_1 guarding at level $\min(l, i)$, let e' denote e_1 and if $[e'; e_2)$ is of type $[e'; e_2]$ and contains only climbing points then we make p_1 climbing at level i else we make p_1 waiting at level i. Let c'_l be the points in c_l which was not moved in the previous movement of points. If $|c'_l| < c$ make c'_l helping at level h. If e' is e_1 then call fix(l). Delete e, call fix(h), fix(i) and rebalance-above(1).

Rebalance-below(i) For each level $l = 0, \ldots, i$ we perform a shift-up(l) while $c < c_l$.

Rebalance-above(i) For each level $l = i, \ldots, m-1$ we perform shift-down($l+1$) while $c_l < -c$.

3 Memory management

We will now deal with the memory layout of the data structure. We will put the blocks in the order $B_0, \ldots, B_m$, where block B_i further has its dictionaries in the order $D_i, A_i, R_i, W_i, H_i, C_i$ and G_i, see Figure 1. Block B_m grows and shrinks to the right when elements are inserted and deleted from the working set dictionary.

The D_i structure is not a moveable dictionary as the other structures in a block are, it is simply an array of $w_i = d2^{i+k}$ elements which we use to encode the size of each of the structures A_i, R_i, W_i, H_i, C_i and G_i along with their own auxiliary data, as they are not implicit and need to remember $\mathcal{O}(2^{i+k})$ bits which we store here. As each of the moveable dictionaries in B_i have size $\mathcal{O}(2^{2^{i+k}})$ we need to encode numbers of $\mathcal{O}(2^{i+k})$ bits in D_i.

We now describe the memory management concerning the movement, insertion and deletion of elements from the working-set dictionary. First notice that the methods find, predecessor and successor do not change the working-set dictionary, and layout in memory. Also the methods shift-down, search, rebalance-below and rebalance-above only calls other methods, hence their memory management is handled by the methods they call. The only methods where actual memory management comes into play are in insert, shift-up, fix, move-down and delete. We will now describe two methods internal-movement – which handles movement inside a single block/level – and external-movement – which handles movement across different blocks/levels. Together these two methods handle all memory management.

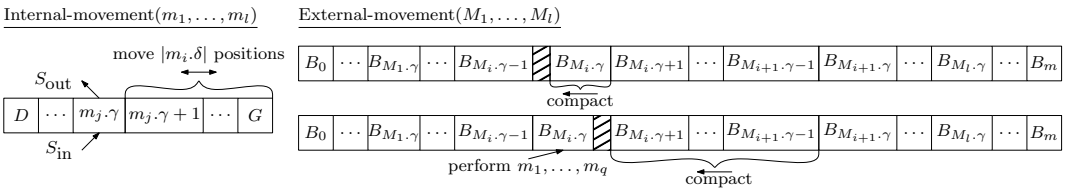


Figure 2 (Left) Memory movement of internal-movement inside of a block B_i. (Right) Memory movement of external-movement across multiple blocks $B_{M_1.\gamma}, \ldots, B_{M_l.\gamma}$.

Internal-movement$(m_1, \ldots, m_l)$ Internal-movement in level i takes a list of *internal moves* $m_1, \ldots, m_l$ to be performed on block B_i, where $l = \mathcal{O}(1)$ and move m_j consists of:

- the index $\gamma = D_i, A_i, R_i, W_i, H_i, C_i, G_i$ of the dictionary to change, where we assume[4] that $m_j.\gamma \leq m_h.\gamma$, for $j \leq h$,
- the set of elements S_{in} to put into γ, where $|S_{\text{in}}| = \mathcal{O}(1)$,
- the set of elements S_{out} to take out of γ, where $|S_{\text{out}}| = \mathcal{O}(1)$ and
- the total size difference $\delta = |S_{\text{in}}| - |S_{\text{out}}|$ of γ after the move.

For $j = 1, \ldots, l$ do: if $m_j.\delta < 0$ then remove S_{out} from γ, insert S_{in} into γ and move $\gamma+1, \ldots, G$ left $|m_j.\delta|$ positions, where we move them in the order $\gamma+1, \ldots, G$. If $m_j.\delta > 0$ then move $\gamma+1, \ldots, G$ right $m_j.\delta$ positions, where we move them in the order $G, \ldots, \gamma+1$, remove S_{out} from γ and insert S_{in} into γ. See Figure 2.

It takes $\mathcal{O}(\log(2^{2^{i+k}})) = \mathcal{O}(2^{i+k})$ time and $\mathcal{O}(\log_B(2^{2^{i+k}})) = \mathcal{O}(\frac{2^{i+k}}{\log B})$ cache-misses to perform move j. In total all the moves $m_1, \ldots, m_l$ use $\mathcal{O}(2^{i+k})$ time and $\mathcal{O}(\frac{2^{i+k}}{\log B})$ cache-misses, as $l = \mathcal{O}(1)$.

External-movement$(M_1, \ldots, M_l)$ External-movement takes a list of *external moves* $M_1, \ldots, M_l$, where $l = \mathcal{O}(1)$. Move M_j consists of:

- the index $0 \leq \gamma \leq m$ of the block/level to perform the internal moves $m_1, \ldots, m_q$ on, where $M_j.\gamma < M_h.\gamma$ for $j < h$,
- the list of internal moves $m_1, \ldots, m_q$ to perform on block γ, where $q = \mathcal{O}(1)$, and
- the total size difference $\Delta = \sum_{h=1}^{q} m_h.\delta$ of block γ after all the internal moves $m_1, \ldots, m_q$ have been performed.

Let $\overline{\Delta} = \sum_{i=1}^{l} M_i.\Delta$ be the total size change of the dictionary after the external-moves have been performed. If $\overline{\Delta} = 0$ then we let $\gamma_{\text{end}} = M_l.\gamma$ else we let $\gamma_{\text{end}} = m$. Let $p_{\text{end}} = \sum_{j=0}^{\gamma_{\text{end}}} |B_j| + \overline{\Delta}$ be the last address of the right most block that we need to alter. Let $s_1, \ldots, s_k$ be the sublist of the indexes $\{1, \ldots, l\}$ where $M_{s_i}.\Delta \leq 0$ for $i = 1, \ldots, k$. Let $a_1, \ldots, a_h$ be the sublist of the indexes $\{1, \ldots, l\}$ where $M_{a_i}.\Delta > 0$ for $i = 1, \ldots, h$.

We first perform all the internal moves of each of the external moves $M_{s_1}, \ldots, M_{s_k}$. Then we compact all the blocks with index i where $M_1.\gamma \leq i \leq \gamma_{\text{end}}$ so the rightmost block ends at position p_{end}. Finally for each external move M_{a_i} for $i = 1, \ldots, h$: move $B_{M_{a_i}.\gamma}$ left so it aligns with $B_{M_{a_i}.\gamma-1}$ and perform all the internal moves of M_{a_i}, then compact the blocks $B_{M_{a_i}.\gamma+1}, \ldots, B_{M_{a_{i+1}}.\gamma-1}$ at the left end so they align with block $B_{M_{a_i}.\gamma}$.

It takes $\mathcal{O}\left(l \log\left(2^{2^{i+k}}\right)\right) = \mathcal{O}\left(l 2^{i+k}\right)$ time and $\mathcal{O}\left(l \log_B\left(2^{2^{i+k}}\right)\right) = \mathcal{O}\left(l \frac{2^{i+k}}{\log B}\right)$ cache-misses to perform the internal moves on level i. In total all the external moves $M_1, \ldots, M_l$ use

[4] We will misuse notation and let $\gamma+1$ denote the next in the total order D, A, R, W, H, C, G. We will also compare $m_j.\gamma$ and $m_h.\gamma$ with $\leq$ in this order.

$\mathcal{O}(2^{\gamma_{\text{end}}+k})$ time and $\mathcal{O}\left(\frac{2^{\gamma_{\text{end}}+k}}{\log B}\right)$ cache-misses, as the external move at level γ_{end} dominates the rest and $l = \mathcal{O}(1)$.

3.1 Memory management in updates of intervals

With the above two methods we can perform the memory management when updating the intervals in Section 2.3: Whenever an element moves around, is deleted or inserted, it is simply put in one or two internal moves. All internal moves in a single block/level are grouped into one external move. Since all updates of intervals only move around a constant number of elements, the requirements for internal/external-movement that $l = \mathcal{O}(1)$ and $q = \mathcal{O}(1)$ are fulfilled. From the above time and cache bounds for the memory management the bounds in Theorem 1 follows.

References

1 Mihai Bădoiu, Richard Cole, Erik D. Demaine, and John Iacono. A unified access bound on comparison-based dynamic dictionaries. *Theoretical Computer Science*, 382(2):86–96, 2007.

2 Prosenjit Bose, Karim Douïeb, and Stefan Langerman. Dynamic optimality for skip lists and B-trees. In *Proc. 19th Annual ACM-SIAM Symposium on Discrete algorithms*, pages 1106–1114. SIAM, 2008.

3 Prosenjit Bose, John Howat, and Pat Morin. A distribution-sensitive dictionary with low space overhead. In *Proc. 11th International Symposium on Algorithms and Data Structures*, volume 5664 of *LNCS*, pages 110–118. Springer-Verlag, 2009.

4 Gerth Brodal, Casper Kejlberg-Rasmussen, and Jakob Truelsen. A cache-oblivious implicit dictionary with the working set property. In *Proc. 12th International Symposium on Algorithms and Data Structures*, volume 6507 of *LNCS*, pages 37–48. Springer-Verlag, 2010.

5 G. Franceschini, R. Grossi, J.I. Munro, and L. Pagli. Implicit B-trees: New results for the dictionary problem. In *Proc. 43rd Annual IEEE Symposium on Foundations of Computer Science*, pages 145–154, 2002.

6 Gianni Franceschini and Roberto Grossi. Optimal worst-case operations for implicit cache-oblivious search trees. In *Proc. 8th International Workshop on Algorithms and Data Structures*, volume 2748 of *LNCS*, pages 114–126. Springer-Verlag, 2003.

7 Gianni Franceschini and Roberto Grossi. Optimal implicit dictionaries over unbounded universes. *Theory of Computing Systems*, 39:321–345, 2006.

8 Matteo Frigo, Charles Eric Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proc. 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 285–297. IEEE, 1999.

9 John Iacono. Alternatives to splay trees with $\mathcal{O}(\log n)$ worst-case access times. In *Proc. 12th Annual ACM-SIAM Symposium on Discrete algorithms*, pages 516–522. SIAM, 2001.

10 James Ian Munro. An implicit data structure supporting insertion, deletion, and search in $\mathcal{O}(\log^2 n)$ time. *Journal of Computer and System Sciences*, 33(1):66–74, 1986.

11 James Ian Munro and Hendra Suwanda. Implicit data structures for fast search and update. *Journal of Computer and System Sciences*, 21(2):236–250, 1980.

12 Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.

Chernoff-Hoeffding Bounds for Markov Chains: Generalized and Simplified

Kai-Min Chung[1], Henry Lam[2], Zhenming Liu[3], and Michael Mitzenmacher[4]

1 **Computer Science Department, Cornell University. Supported by a Simons Foundation Fellowship.**
chung@cs.cornell.edu
2 **Mathematics and Statistics Department, Boston University.**
khlam@bu.edu
3 **Harvard School of Engineering and Applied Sciences. Supported by NSF grant CCF-0915922.**
zliu@fas.harvard.edu
4 **Harvard School of Engineering and Applied Sciences. Supported in part by NSF grants CCF-0915922 and IIS-0964473.**
michaelm@eecs.harvard.edu

Abstract

We prove the first Chernoff-Hoeffding bounds for *general* (*nonreversible*) finite-state Markov chains based on the standard L_1 (variation distance) *mixing-time* of the chain. Specifically, consider an ergodic Markov chain M and a weight function $f : [n] \to [0, 1]$ on the state space $[n]$ of M with mean $\mu \triangleq \mathrm{E}_{v\leftarrow\pi}[f(v)]$, where π is the stationary distribution of M. A t-step random walk $(v_1, \ldots, v_t)$ on M starting from the stationary distribution π has expected total weight $\mathrm{E}[X] = \mu t$, where $X \triangleq \sum_{i=1}^{t} f(v_i)$. Let T be the L_1 mixing-time of M. We show that the probability of X deviating from its mean by a multiplicative factor of δ, i.e., $\Pr[|X - \mu t| \geq \delta\mu t]$, is at most $\exp(-\Omega\left(\delta^2\mu t/T\right))$ for $0 \leq \delta \leq 1$, and $\exp(-\Omega(\delta\mu t/T))$ for $\delta > 1$. In fact, the bounds hold even if the weight functions f_i's for $i \in [t]$ are distinct, provided that all of them have the same mean μ.

We also obtain a simplified proof for the Chernoff-Hoeffding bounds based on the *spectral expansion* λ of M, which is the square root of the second largest eigenvalue (in absolute value) of $M\tilde{M}$, where $\tilde{M}$ is the time-reversal Markov chain of M. We show that the probability $\Pr[|X - \mu t| \geq \delta\mu t]$ is at most $\exp(-\Omega\left(\delta^2(1-\lambda)\mu t\right))$ for $0 \leq \delta \leq 1$, and $\exp(-\Omega(\delta(1-\lambda)\mu t))$ for $\delta > 1$.

Both of our results extend to continuous-time Markov chains, and to the case where the walk starts from an arbitrary distribution x, at a price of a multiplicative factor depending on the distribution x in the concentration bounds.

1998 ACM Subject Classification G.3 Probability and Statistics

Keywords and phrases probabilistic analysis, tail bounds, Markov chains

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.124

1 Introduction

In this work, we establish large deviation bounds for random walks on general (irreversible) finite state Markov chains based on mixing properties of the chain in both discrete and continuous time settings. To introduce our results we focus on the discrete time setting, which we now describe.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 124–135

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

Let M be an ergodic Markov chain with finite state space $V = [n]$ and stationary distribution π. Let $(v_1, \ldots, v_t)$ denote a t-step random walk on M starting from a distribution φ on V. For every $i \in [t]$, let $f_i : V \to [0,1]$ be a weight function at step i so that $\mathrm{E}_{v \leftarrow \pi}[f_i(v)] = \mu > 0$ for all i. Define the total weight of the walk $(v_1, \ldots, v_t)$ by $X \triangleq \sum_{i=1}^{t} f_i(v_i)$. The expected total weight of the random walk $(v_1, \ldots, v_t)$ is $\mathrm{E}[\frac{1}{t}X] \approx \mu$ as $t \to \infty$.

When the v_i's are drawn independently according to the stationary distribution π, a standard Chernoff-Hoeffding bound says that

$$\Pr\left[|X - \mu t| \geq \delta\mu t\right] \leq \begin{cases} e^{-\Omega\left(\delta^2 \mu t\right)} & \text{for } 0 \leq \delta \leq 1, \\ e^{-\Omega(\delta \mu t)} & \text{for } \delta > 1. \end{cases}$$

However, when $(v_1, \ldots, v_t)$ is a random walk on a Markov chain M, it is known that the concentration bounds depend inherently on the mixing properties of M, that is the speed at which a random walk converges toward its stationary distribution.

Variants of Chernoff-Hoeffding bounds for random walk on Markov chains have been studied in several fields with various motivations [5, 10, 11, 12, 17, 16, 7]. For instance, these bounds are linked to the performance of Markov chain Monte Carlo integration techniques [11, 9]. They have also been applied to various online learning problem [15], testing properties of a given graph [6], leader election problems [10], analyzing the structure of the social networks [2, 13], understanding the performance of data structures [4], and computational complexity [7]. Improving such bounds is therefore of general interest.

We improve on previous work in two ways. First, all the existing deviation bounds, as far as we know, are based on the *spectral expansion* $\lambda(M)$ of the chain M. This spectral expansion $\lambda(M)$ characterizes how much M can stretch vectors in $\mathbf{R}^n$ under a normed space defined by the stationary distribution π, which coincides with the second largest absolute eigenvalue of M when M is reversible. (A formal definition is deferred to Section 2.) The most general result for Markov chains in this form (see, e.g. [12, 16]) is

$$\Pr\left[|X - \mu t| \geq \delta\mu t\right] \leq \begin{cases} \|\varphi\|_\pi e^{-\Omega\left((1-\lambda)\delta^2 \mu t\right)} & \text{for } 0 \leq \delta \leq 1, \\ \|\varphi\|_\pi e^{-\Omega((1-\lambda)\delta \mu t)} & \text{for } \delta > 1. \end{cases} \tag{1}$$

where φ is an arbitrary initial distribution and $\|\cdot\|_\pi$ is the π-norm (which we define formally later).

However, for general irreversible Markov chains, the spectral expansion λ does not directly characterize the mixing time of a chain and thus may not be a suitable parameter for such bounds. A Markov chain M could mix rapidly, but have a spectral expansion λ close to 1, in which case Eq. (1) does not yield meaningful bound. In fact there is a way to modify any given Markov chain M so that the modified Markov chain M' has (asymptotically) the same mixing-time as M, but the spectral expansion of M' equals 1 (Appendix A gives a detailed construction). It is therefore natural to seek a Chernoff-type bound for Markov chains directly parameterized by the chain's mixing time T.

Second, most previous analyses for deviation bounds such as Eq. (1) are based on *non-elementary* methods such as perturbation theory [5, 12, 11, 17]. Kahale [10] and Healy [7] provided two elementary proofs for reversible chains, but their results yield weaker bounds than those in Eq. (1). Recently, Wagner [16] provided another elementary proof for reversible chains matching the form in Eq. (1). Together with the technique of "reversiblization" [3, 12], Wagner's analysis can be generalized to irreversible chains. However, his use of decoupling on the linear projections outright arguably leads to a loss of insight; here we provide an approach based on directly tracing the corresponding sequence of linear projections, in the spirit of [7]. This more elementary approach allows us to tackle both reversible and irreversible chains in a unified manner that avoids the use of "reversiblization".

As we describe below, we prove a Chernoff-type bound for general irreversible Markov chains with general weight functions f_i based on the standard L_1 (variation distance) mixing time of the chain, using elementary techniques based on extending ideas from [7]. The exponents of our bounds are tight up to a constant factor. As far as we know, this is the first result that shows that the mixing time is sufficient to yield these types of concentration bounds for random walks on Markov chains. Along the way we provide a unified proof for (1) for both reversible and irreversible chains based only on elementary analysis. This proof may be of interest in its own right.

2 Preliminaries

Throughout this paper we shall refer M as the discrete time Markov chain under consideration. Depending on the context, M shall be interpreted as either the chain itself or the corresponding transition matrix (i.e. it is an n by n matrix such that $M_{i,j}$ represents the probability a walk at state i will move to state j in the next step). For the continuous time counterpart, we write Λ as the generator of the chain and let $M(t) = e^{t\Lambda}$, which represents the transition probability matrix from t_0 to $t_0 + t$ for an arbitrary t_0.

Let u and w be two distributions over the state space $\mathbf{V}$. The *total variation* distance between u and w is $\|u - w\|_{TV} = \max_{A \subseteq \mathbf{V}} \left|\sum_{i \in A} u_i - \sum_{i \in A} w_i\right| = \frac{1}{2}||u - w||_1$.

Let $\epsilon > 0$. The mixing time of a *discrete time* Markov chain M is $T(\epsilon) = \min\{t : \max_x \|xM^t - \pi\|_{TV} \leq \epsilon\}$, where x is an arbitrary initial distribution. The mixing time of a *continuous time* Markov chain specified by the generator Λ is $T(\epsilon) = \min\{t : \max_x \|xM(t) - \pi\|_{TV} \leq \epsilon\}$, where $M(t) = e^{\Lambda t}$.

We next define an inner product space specified by the stationary distribution π:

▶ **Definition 1** (Inner product under π-kernel)**.** Let M be an ergodic Markov chain with state space $[n]$ and π be its stationary distribution. Let u and v be two vectors in R^n. The *inner product under the π-kernel* is $\langle u, v\rangle_\pi = \sum_{x \in [n]} \frac{u_i v_i}{\pi(i)}$.

We may verify that $\langle \cdot, \cdot \rangle_\pi$ indeed forms an inner product space by checking it is symmetric, linear in the first argument, and positive definite. The π-norm of a vector u in R^n is $\|u\|_\pi = \sqrt{\langle u, u\rangle_\pi}$. Note that $\|\pi\|_\pi = 1$. For a vector $x \in R^n$, we write $x^{\|} = \langle x, \pi\rangle_\pi \pi$ for its component along the direction of π and $x^\perp = x - x^{\|}$ for its component perpendicular to π.

We next define the *spectral norm* of a transition matrix.

▶ **Definition 2** (Spectral norm)**.** Let M the transition matrix of an ergodic Markov chain. Define the spectral norm of M as $\lambda(M) = \max_{\langle x, \pi\rangle_\pi = 0} \frac{\|xM\|_\pi}{\|x\|_\pi}$.

When M is clear from the context, we shall simply write λ for $\lambda(M)$. We shall also refer $1 - \lambda(M)$ as the *spectral gap* of the chain M. In the case when M is reversible, $\lambda(M)$ coincides with the second largest eigenvalue of M (the largest eigenvalue of M is always 1). However, when M is irreversible, such relation does not hold (one hint to realize that the eigenvalues of M for an irreversible chain can be complex, and the notion of being the second largest may not even be well defined). Nevertheless, we can still connect $\lambda(M)$ with an eigenvalue of a matrix related to M. Specifically, let $\tilde{M}$ be the time reversal of M: $\tilde{M}(x, y) = \frac{\pi(y)M(y,x)}{\pi(x)}$. The *multiplicative reversiblization* $R(M)$ of M is $R(M) \equiv M\tilde{M}$. The value of $\lambda(M)$ then coincides with the square root of the second largest eigenvalue of $R(M)$, i.e. $\lambda(M) = \sqrt{\lambda(R(M))}$. Finally, notice that the stationary distribution of M, $\tilde{M}$, and R are all the same. These facts can be found in [3].

3 Chernoff-Hoeffding Bounds for Discrete Time Markov Chains

We now present our main result formally.

▶ **Theorem 3.** *Let M be an ergodic Markov chain with state space $[n]$ and stationary distribution π. Let $T = T(\epsilon)$ be its ε-mixing time for $\varepsilon \leq 1/8$. Let $(V_1, \ldots, V_t)$ denote a t-step random walk on M starting from an initial distribution φ on $[n]$, i.e., $V_1 \leftarrow \varphi$. For every $i \in [t]$, let $f_i : [n] \to [0,1]$ be a weight function at step i such that the expected weight $\mathrm{E}_{v \leftarrow \pi}[f_i(v)] = \mu$ for all i. Define the total weight of the walk $(V_1, \ldots, V_t)$ by $X \triangleq \sum_{i=1}^{t} f_i(V_i)$. There exists some constant c (which is independent of μ, δ and ϵ) such that*

1. $\Pr[X \geq (1+\delta)\mu t] \leq \begin{cases} c\|\varphi\|_\pi \exp\left(-\delta^2\mu t/(72T)\right) & \text{for } 0 \leq \delta \leq 1 \\ c\|\varphi\|_\pi \exp\left(-\delta\mu t/(72T)\right) & \text{for } \delta > 1 \end{cases}$
2. $\Pr[X \leq (1-\delta)\mu t] \leq c\|\varphi\|_\pi \exp\left(-\delta^2\mu t/(72T)\right)$ *for* $0 \leq \delta \leq 1$

Before we continue our analysis, we remark on some aspects of the result.

Optimality of the bound The bound given in Theorem 3 is optimal among all bounds based on the mixing time of the Markov chain, in the sense that for any given T and constant ε, one can find a δ, a family of functions $\{f_i : V \to [0,1]\}$, and a Markov chain with mixing time $T(\varepsilon) = T$ that has deviation probabilities matching the exponents displayed in Theorem 3, up to a constant factor. In this regard, the form of our dependency on T is tight for constant ε. For example, consider the following Markov chain:

- The chain consists of 2 states s_1 and s_2.
- At any time step, with probability p the random walk jumps to the other state and with probability $1-p$ it stays in its current state, where p is determined below.
- for all f_i, we have $f_i(s_1) = 1$ and $f_i(s_2) = 0$.

Notice that the stationary distribution is uniform and $T(\epsilon) = \Theta(1/p)$ when ϵ is a constant. Thus, we shall set $p = \Theta(1/T)$ so that the mixing-time $T(\varepsilon) = T$. Let us consider a walk starting from s_1 for sufficiently large length t. The probability that the walk stays entirely in s_1 up to time t is $(1-p)^t \approx e^{-tp} = \exp(-\Theta(t/T))$. In other words, for $\delta = 1$ we have $\Pr[X \geq (1+\delta)\mu t] = \Pr[X \geq t] = \Pr[\text{the walk stays entirely in } s_1] = \exp(-\Theta(t/T(\epsilon)))$. This matches the first bound in Theorem 3 asymptotically, up to a constant factor in the exponent. The second bound can be matched similarly by switching the values of $f_i(\cdot)$ on s_1 and s_2. Finally, we remark that this example only works for $\epsilon = \Omega(1)$, which is how mixing times appear in the usual contexts. It remains open, though, whether our bounds are still optimal when $\epsilon = o(1)$.

Dependency on the threshold ϵ of the mixing time Note that the dependence of ϵ only lies on $T(\epsilon)$. Since $T(\epsilon)$ is non-decreasing in ϵ, it is obvious that $\epsilon = 1/8$ gives the best bound in the setting of Theorem 3. In fact, a more general form of our bound, as will be seen along our derivation later, replaces $1/72$ in the exponent by a factor $(1-\sqrt{2\epsilon})/36$. Hence the optimal choice of ϵ is the maximizer of $(1-\sqrt{2\epsilon})/T(\epsilon)$ (with $\epsilon < 1/2$), which differs for different Markov chains. Such formulation seems to offer incremental improvement and so we choose to focus on the form in Theorem 3.

Comparison with spectral expansion based Chernoff bound The bound given in Theorem 3 is *not* always stronger than spectral expansion based Chernoff bounds (1) that is presented in, for example, Lezaud [12] and Wagner [16]. Consider, for instance, a random constant degree regular graph G. One can see that the spectral gap of the Markov chain induced by a random walk over G is a constant with high probability. On the other

hand, the mixing time of the chain is at least $\Omega(\log n)$ because the diameter of a constant degree graph is at least $\Omega(\log n)$. Lezaud [12] or Wagner [16] gives us a concentration bound $\Pr[X \geq (1+\epsilon)\mu t] \leq c\|\varphi\|_\pi \exp\left(-\Theta(\delta^2\mu t)\right)$ when $\delta < 1$ while Theorem 3 gives us $\Pr[X \geq (1+\epsilon)\mu t] \leq c\|\varphi\|_\pi \exp\left(-\Theta(\delta^2\mu t/(\log n))\right)$.

Comparison with a union bound Assuming the spectral expansion based Chernoff bound in Lezaud [12] and Wagner [16], there is a simpler analysis to yield a mixing time based bound in a similar but weaker form than Theorem 3: we first divide the random walk $(V_1, ..., V_t)$ into $T(\epsilon)$ groups for a sufficiently small ϵ such that the ith group consists of the sub-walk $V_i, V_{i+T(\epsilon)}, V_{i+2T(\epsilon)}, ...$ The walk in each group is then governed by the Markov chain $M^{T(\epsilon)}$. This Markov chain has unit mixing time and as a result, its spectral expansion can be bounded by a constant (by using our Claim 3.1 below). Together with a union bound across different groups, we obtain

$$\begin{aligned}
&1.\ \Pr[X \geq (1+\delta)\mu t] \quad \leq \quad \begin{cases} cT\|\varphi\|_\pi \exp\left(-\delta^2\mu t/(72T)\right) & \text{for } 0 \leq \delta \leq 1 \\ cT\|\varphi\|_\pi \exp\left(-\delta\mu t/(72T)\right) & \text{for } \delta > 1 \end{cases} \\
&2.\ \Pr[X \leq (1-\delta)\mu t] \quad \leq \quad cT\|\varphi\|_\pi \exp\left(-\delta^2\mu t/(72T)\right) \qquad \text{for } 0 \leq \delta \leq 1
\end{aligned} \tag{2}$$

Theorem 3 shaves off the extra leading factors of T in these inequalities, which has significant implications. For example, Eq. (2) requires the walk to be at least $\Omega(T \log T)$, while our bounds address walk lengths between T and $T \log T$. Our tighter bound further can become important when we need a tighter polynomial tail bound.

As a specific example, saving the factor of T becomes significant when we generalize these bounds to continuous-time chains using the discretization strategy in Fill [3] and Lezaud [12]. The strategy is to apply known discrete time bound on the discretized continuous time chain, say in a scale of b units of time, followed by taking limit as $b \to 0$ to yield the corresponding continuous time bound. Using this to obtain a continuous analog of Eq. (2) does not work, since under the b-scaled discretization the mixing time becomes T/b, which implies that the leading factor in Eq. (2) goes to infinity in the limit as $b \to 0$.

We now proceed to prove Theorem 3.

Proof. (of Theorem 3) We partition the walk $V_1, ..., V_t$ into $T = T(\epsilon)$ subgroups so that the i-th sub-walk consists of the steps $(V_i, V_{i+T}, ...)$. These sub-walks can be viewed as generated from Markov chain $N \triangleq M^T$. Also, denote $X^{(i)} \triangleq \sum_{0 \leq j \leq t/T} f_{i+jT}(V_{i+jT})$ as the total weight for each sub-walk and $\bar{X} = \sum_{i=1}^{T} X^{(i)}/T$ as the average total weight.

Next, we follow Hoeffding's approach [8] to cope with the correlation among the $X^{(i)}$. To start,

$$\Pr[X \geq (1+\delta)\mu t] = \Pr\left[\bar{X} \geq (1+\delta)\frac{\mu t}{T}\right] \leq \frac{\mathrm{E}[e^{r\bar{X}}]}{e^{r(1+\delta)\mu t/T}}. \tag{3}$$

Now noting that $\exp(\cdot)$ is a convex function, we use Jensen's inequality to obtain

$$\mathrm{E}[e^{r\bar{X}}] \leq \sum_{i \leq T} \frac{1}{T}\mathrm{E}[e^{rX^{(i)}}]. \tag{4}$$

We shall focus on giving an upper bound on $\mathrm{E}[e^{rX^{(i)}}]$. This requires two steps:

- First, we show the chain N has a constant spectral gap based on the fact that it takes one step to mix.
- Second, we appy a bound on the moment generating function of $X^{(k)}$ using its spectral expansion.

Specifically, we shall prove the following claims, whose proofs will be deferred to the next two subsections.

▶ Claim 3.1. Let M be a general ergodic Markov chain with ϵ-mixing time $T(\epsilon)$. We have $\lambda(M^{T(\epsilon)}) \leq \sqrt{2\epsilon}$.

▶ Claim 3.2. Let M be an ergodic Markov chain with state space $[n]$, stationary distribution π, and spectral expansion $\lambda = \lambda(M)$. Let $(V_1, \ldots, V_t)$ denote a t-step random walk on M starting from an initial distribution φ on $[n]$, i.e., $V_1 \leftarrow \varphi$. For every $i \in [t]$, let $f_i : [n] \to [0,1]$ be a weight function at step i such that the expected weight $\mathrm{E}_{v \leftarrow \pi}[f_i(v)] = \mu$ for all i. Define the total weight of the walk $(V_1, \ldots, V_t)$ by $X \triangleq \sum_{i=1}^{t} f_i(V_i)$. There exists some constant c and a parameter $r > 0$ that depends only on λ and δ such that

1. $\dfrac{\mathrm{E}[e^{rX}]}{e^{r(1+\delta)\mu t}} \leq \begin{cases} c\|\varphi\|_\pi \exp\left(-\delta^2\ (1-\lambda)\mu t/36\right) & \text{for } 0 \leq \delta \leq 1 \\ c\|\varphi\|_\pi \exp\left(-\delta(1-\lambda)\mu t/36\right) & \text{for } \delta > 1. \end{cases}$
2. $\dfrac{\mathrm{E}[e^{-rX}]}{e^{-r(1-\delta)\mu t}} \leq c\|\varphi\|_\pi \exp\left(-\delta^2(1-\lambda)\mu t/36\right) \quad \text{for } 0 \leq \delta \leq 1.$

Claim 3.1 gives a bound on the spectral expansion of each sub-walk $X^{(i)}$, utilizing the fact that they have unit mixing times. Claim 3.2 is a spectral version of Chernoff bounds for Markov chains. As stated previously, while similar results exist, we provide our own elementary proof of claim 3.2, both for completeness and because it may be of independent interest.

We now continue the proof assuming these two claims. Using Claim 3.1, we know $\lambda(N) \leq \frac{1}{2}$. Next, by Claim 3.2, for the i-th sub-walk, we have

$$\frac{\mathrm{E}[e^{rX^{(i)}}]}{e^{r(1+\delta)\mu t/T}} \leq \begin{cases} c\|\varphi M^i\|_\pi \exp\left(-\delta^2\mu t/(72T)\right) & \text{for } 0 \leq \delta \leq 1 \\ c\|\varphi M^i\|_\pi \exp\left(-\delta\mu t/(72T)\right) & \text{for } \delta > 1 \end{cases} \tag{5}$$

for an appropriately chosen r (which depends only on λ and δ and hence the same for all i). Note that M^i arises because $X^{(i)}$ starts from the distribution φM^i. On the other hand, notice that $\|\varphi M^i\|_\pi^2 = \|\varphi^{\|} M^i\|_\pi^2 + \|\varphi^{\perp} M^i\|_\pi^2 \leq \|\varphi^{\|}\|_\pi^2 + \lambda^2(M^i)\|\varphi^{\perp}\|_\pi^2 \leq \|\varphi\|_\pi^2$ (by using Lemma 5), or in other words $\|\varphi M^i\|_\pi \leq \|\varphi\|_\pi$. Together with (3) and (4), we obtain

$$\Pr[X \geq (1+\delta)\mu t] \leq \begin{cases} c\|\varphi\|_\pi \exp\left(-\delta^2\mu t/(72T)\right) & \text{for } 0 \leq \delta \leq 1 \\ c\|\varphi\|_\pi \exp\left(-\delta\mu t/(72T)\right) & \text{for } \delta > 1 \end{cases}$$

This proves the first half of the theorem. The second case can be proved in a similar manner, namely that

$$\Pr[X \leq (1-\delta)\mu t] \leq \frac{\mathrm{E}[e^{-r\bar{X}}]}{e^{-r(1-\delta)\mu t/T}} \leq \sum_{k=1}^{T} \frac{1}{T} \frac{\mathrm{E}[e^{-rX^{(k)}}]}{e^{-r(1-\delta)\mu t/T}} \leq c\|\varphi\|_\pi \exp\left(-\delta^2\mu t/(72T)\right)$$

again by Jensen's inequality applied to $\exp(\cdot)$.

◀

3.1 Mixing Time v.s. Spectral Expansion

In this subsection we prove Claim 3.1. We remark that Sinclair [14] presents a similar result for reversible Markov chains: for every parameter $\varepsilon \in (0,1)$,

$$\frac{1}{2}\frac{\lambda(M)}{1-\lambda(M)} \log\frac{1}{2\varepsilon} \leq T(\varepsilon), \tag{6}$$

where $T(\varepsilon)$ is the ε-mixing-time of M. However, in general it is impossible to get a bound on $\lambda(M)$ based on mixing time information for general irreversible chains because a chain M

can have $\lambda(M) = 1$ but the ε-mixing-time of M is, say, $T(\varepsilon) = 2$ for some constant ϵ (and $\lambda(M^2) \ll 1$).

In light of this issue, our proof of Claim 3.1 depends crucially on the fact that $M^{T(\varepsilon)}$ has mixing time 1, which, as we shall see, translates to a bound on its spectral expansion that holds regardless of reversibility. We need the following result on reversible Makrov chains, which is stronger result than Eq. (6) from [14].

▶ **Lemma 4.** *Let $0 < \varepsilon \leq 1/2$ be a parameter. Let M be an ergodic reversible Markov chain with ε-mixing time $T(\varepsilon)$ and spectral expansion $\lambda(M)$. It holds that $\lambda(M) \leq (2\varepsilon)^{1/T(\varepsilon)}$.*

We remark that it appears possible to prove Lemma 4 by adopting an analysis similar to Aldous' [1], who addressed the continuous time case. We present an alternative proof that is arguably simpler; in particular, our proof does not use the spectral representation theorem as used in [1] and does not involve arguments that take the number of steps to infinity.

Proof. (of Lemma 4) Recall that for an ergodic reversible Markov chain M, it holds that $\lambda(M^t) = \lambda^t(M)$ for every $t \in \mathbb{N}$. Hence, it suffices to show that $\lambda(M^{T(\epsilon)}) \leq 2\epsilon$. Also, recall that $\lambda(M^{T(\epsilon)})$ is simply the second largest eigenvalue (in absolute value) of $M^{T(\epsilon)}$. Let v be the corresponding eigenvector, i.e. v satisfies $vM^{T(\epsilon)} = \lambda(M^{T(\epsilon)})v$. Since M is reversible, the entries of v are real-valued. Also, notice that v is a left eigenvector of M while $(1, 1, ..., 1)^T$ is a right eigenvector of M (using the fact that each row of M sums to one). Furthermore, v and $(1, ..., 1)^T$ do not share the same eigenvalue. So we have $\langle v, (1, ..., 1)^T \rangle = 0$, i.e. $\sum_i v_i = 0$. Therefore, by scaling v, we can assume w.l.o.g. that $x \triangleq v + \pi$ is a distribution. We have the following claim.

▶ Claim 3.3. Let x be an arbitrary initial distribution. Let M be an ergodic Markov chain with stationary distribution π and mixing time $T(\epsilon)$. We have $\|xM^{T(\epsilon)} - \pi\|_{TV} \leq 2\epsilon\|x - \pi\|_{TV}$.

The key idea for proving this claim is to split the difference $x-\pi$ into positive and negative components, and analyze $\|xM^{T(\epsilon)} - \pi\|_{TV} = \|(x - \pi)M^{T(\epsilon)}\|_{TV}$ using these components together with a rescaling to transform the components into probability distributions and then invoke the definition of mixing time. Details are available in the full version of this paper.

By Claim 3.3, $\|xM^{T(\varepsilon)} - \pi\|_{TV} \leq 2\varepsilon\|x - \pi\|_{TV}$, i.e. $\|xM^{T(\varepsilon)} - \pi\|_1 \leq 2\varepsilon\|x - \pi\|_1$. Observing that $(xM^{T(\varepsilon)} - \pi)$ and $(x - \pi)$ are simply $\lambda(M^{T(\varepsilon)})v$ and v, the above inequality means $\lambda(M^{T(\varepsilon)})\|v\|_1 \leq 2\varepsilon\|v\|_1$, which implies $\lambda(M^{T(\varepsilon)}) \leq 2\varepsilon$, as desired. ◀

Proof. (of Claim 3.1) The idea is to reduce to the reversible case by considering the reversiblization of $M^{T(\varepsilon)}$. Let $\tilde{M}^{T(\varepsilon)}$ be the time reversal of $M^{T(\varepsilon)}$, and $R \triangleq M^{T(\varepsilon)}\tilde{M}^{T(\varepsilon)}$ be the reversiblization of $M^{T(\varepsilon)}$. By Claim 3.1, $\lambda(M^{T(\varepsilon)}) = \sqrt{\lambda(R)}$. Let us recall (from Section 2) that M, $M^{T(\varepsilon)}$, and $\tilde{M}^{T(\varepsilon)}$ all share the same stationary distribution π. Next, we claim that the ε-mixing-time of R is 1. This is because $\|\varphi M^{T(\varepsilon)}\tilde{M}^{T(\varepsilon)} - \pi\|_{TV} \leq \|\varphi M^{T(\varepsilon)} - \pi\|_{TV} \leq \varepsilon$, where the second inequality uses the definition of $T(\varepsilon)$ and the first inequality holds since any Markov transition is a contraction mapping: for any Markov transition, say $S = (s(i, j))$, and any vector x, $\|xS\|_1 = \sum_j |\sum_i x_i s(i, j)| \leq \sum_j \sum_i |x_i| s(i, j) = \sum_i |x_i| = \|x\|_1$; putting $x = \varphi M^{T(\varepsilon)} - \pi$ and $S = \tilde{M}^{T(\varepsilon)}$ gives the first inequality. Now, by Lemma 4, $\lambda(R) \leq 2\varepsilon$, and hence $\lambda(M^{T(\varepsilon)}) = \sqrt{\lambda(R)} \leq \sqrt{2\varepsilon}$, as desired. ◀

3.2 Bounding the Moment Generating Function

We now prove Claim 3.2. We focus on the first inequality in the claim; the derivation of the second inequality is similar.

Claim 3.2 leads directly to a spectral version of the Chernoff bound for Markov chains. Lezaud [12] and Wagner [16] give similar results for the case where f_i are the same for all i. The analysis of [16] in particular can be extended to the case where the functions f_i are different. Here we present an alternative analysis and along the way will discuss the merit of our approach compared to the previous proofs.

Recall that we define $X = \sum_{i=1}^{t} f_i(V_i)$. We start with the following observation, which has been used previously [7, 12, 16]:

$$\mathrm{E}[e^{rX}] = \|\varphi P_1 M P_2 ... M P_t\|_1, \tag{7}$$

where the P_i are diagonal matrices with diagonal entries $(P_i)_{j,j} \triangleq e^{rf_i(j)}$ for $j \in [n]$. One can verify this fact by observing that each walk $V_1, \ldots, V_t$ is assigned the corresponding probability in the product of M's with the appropriate weight $e^{r\sum_i f_i(V_i)}$.

For ease of exposition, let us assume P_i are all the same at this moment. Let $P = P_1 = ... = P_t$, then (7) becomes $\|\varphi(PM)^{t-1}P\|_1 = \langle \varphi(PM)^{t-1}P, \pi\rangle_\pi = \langle \varphi(PM)^t, \pi\rangle_\pi = \|\varphi(PM)^t\|_1$ (see Lemma 5 below). Up to this point, our analysis is similar to previous work [5, 12, 7, 16]. Now there are two natural possible ways of bounding $\|\varphi(PM)^t\|_1 = \langle \varphi(PM)^t, \pi\rangle_\pi$.

- **Approach 1. Bounding the spectral norm of the matrix PM.** In this approach, we observe that $\langle \varphi(PM)^t, \pi\rangle_\pi \le \|\varphi\|_\pi \|PM\|_\pi^t$ where $\|PM\|_\pi$ is the operator norm of the matrix PM induced by $\|\cdot\|_\pi$ (see, for example, the proof of Theorem 1 in [16]). This method decouples the effect of each PM as well as the initial distribution. When M is reversible, $\|PM\|_\pi$ can be bounded through Kato's spectral perturbation theory [5, 12, 11]. Alternatively, Wagner [16] tackles the variational description of $\|PM\|_\pi$ directly, using only elementary techniques, whose analysis can be generalized to irreversible chains.
- **Approach 2. Inductively giving a bound for $x(PM)^i$ for all $i \le t$.** In this approach, we do not decouple the product $\varphi(PM)^t$. Instead, we trace the change of the vector $\varphi(PM)^i$ for each $i \le t$. As far as we know, only Healy [7] adopts this approach and his analysis is restricted to regular graphs, where the stationary distribution is uniform. His analysis also does not require perturbation theory.

Our proof here generalizes the second approach to any ergodic chains by only using elementary methods. We believe this analysis is more straightforward for the following reasons. First, directly tracing the change of the vector $\varphi(PM)^i$ for each step keeps the geometric insight that would otherwise be lost in the decoupling analysis as in [12, 16]. Second, our analysis studies both the reversible and irreversible chains in a unified manner. We *do not* use the reversiblization technique to address the case for irreversible chains. While the reversiblization technique is a powerful tool to translate an irreversible Markov chain problem into a reversible chain problem, this technique operates in a blackbox manner; proofs based on this technique do not enable us to directly measure the effect of the operator PM.

We now continue our analysis by using a framework similar to the one presented by Healy [7]. We remind the reader that we no longer assume P_i's are the same. Also, recall that $\mathrm{E}[e^{rX}] = \|\varphi P_1 M P_2 ... M P_t\|_1 = \langle \varphi P_1 M P_2 ... M P_t, \pi\rangle_\pi = \|(\varphi P_1 M P_2 ... M P_t)^\parallel\|_\pi$. Let us briefly review the strategy from [7].

- First, we observe that an arbitrary vector x in $\mathbb{R}^n$ can be decomposed into its *parallel* component (with respect to π) $x^\parallel = \langle x, \pi\rangle\pi$ and the *perpendicular* component $x^\perp = x - x^\parallel$ in the L_π space. This decomposition helps tracing the difference (in terms of the norm) between each pair of $\varphi P_1 M ... P_i M$ and $\varphi P_1 M ... P_{i+1} M$ for $i \le t$, i.e. two consecutive

steps of the random walk. For this purpose, we need to understand the *effects of the linear operators* M and P_i when they are applied to an arbitrary vector.
- Second, after we compute the difference between each pair $xP_1M...P_iM$ and $xP_1M...P_{i+1}M$, we set up a *recursive relation*, the solution of which yields the Chernoff bound.

We now follow this step step framework to prove Claim 3.2

The effects of the M and P_i operators Our way of tracing the vector $\varphi P_1MP_2...MP_t$ relies on the following two lemmas.

▶ **Lemma 5. (The effect of the M operator)** *Let M be an ergodic Markov chain with state space $[n]$, stationary distribution π, and spectral expansion $\lambda = \lambda(M)$. Then*

1. $\pi M = \pi$.
2. *For every vector y with $y \perp \pi$, we have $yM \perp \pi$ and $\|yM\|_\pi \leq \lambda \|y\|_\pi$.*

Note that Lemma 5 is immediate from the definitions of π and λ.

▶ **Lemma 6. (The effect of the P operator)** *Let M be an ergodic Markov chain with state space $[n]$ and stationary distribution π. Let $f : [n] \to [0,1]$ be a weight function with $\mathrm{E}_{v \leftarrow \pi}[f(v)] = \mu$. Let P be a diagonal matrix with diagonal entries $P_{j,j} \triangleq e^{rf(j)}$ for $j \in [n]$, where r is a parameter satisfying $0 \leq r \leq 1/2$. Then*

1. $\|(\pi P)^{\|}\|_\pi \leq 1 + (e^r - 1)\mu$.
2. $\|(\pi P)^{\perp}\|_\pi \leq 2r\sqrt{\mu}$.
3. *For every vector $y \perp \pi$, $\|(yP)^{\|}\|_\pi \leq 2r\sqrt{\mu}\|y\|_\pi$.*
4. *For every vector $y \perp \pi$, $\|(yP)^{\perp}\|_\pi \leq e^r \|y\|_\pi$*

Items 1 and 4 of Lemma 6 state that P can stretch both the perpendicular and parallel components along their original directions moderately. Specifically, a parallel vector is stretched by at most a factor of $(1 + (e^r - 1)\mu) \approx 1 + O(r\mu)$ and a perpendicular vector is stretched by a factor of at most $e^r \approx 1 + O(r)$. (Recall r will be small.) On the other hand, items 2 and 3 of the lemma state that P can create a new perpendicular component from a parallel component and vice versa, but the new component is of a much smaller size compared to the original component (i.e. only of length at most $2r\sqrt{\mu}$ times the original component).

We note that the key improvement of our analysis (which can be found in the full version of this work) over that of Healy [7] stems from items 2 and 3 of Lemma 6. Healy [7] proved a bound with a factor of $(e^r - 1)/2 = O(r)$ for both items for the special case of undirected and regular graphs. Our quantitative improvement to $O(r\sqrt{\mu})$ (which is tight) is the key for us to prove a multiplicative Chernoff bound without any restriction on the spectral expansion of M.

Recursive analysis We now provide a recursive analysis for the terms $xP_1M...MP_i$ for $i \leq t$ based on our understanding of the effects from the linear operators M and P_i. This completes the proof for Claim 3.2.

Sketch of proof of Claim 3.2. First, recall that

$$\mathrm{E}[e^{rX}] = \|(\varphi P_1 MP_2...MP_t)^{\|}\|_\pi = \|(\varphi P_1 MP_2...MP_t M)^{\|}\|_\pi = \left\| \left(\varphi \prod_{i=1}^{t} (P_i M) \right)^{\|} \right\|_\pi$$

where the second equality comes from Lemma 5. Our choice of r is $r = \min\{1/2, \log(1/\lambda)/2, 1 - \sqrt{\lambda}, (1-\lambda)\delta/18\}$. We shall explain how we make such a choice as we walk through our analysis.

We now trace the π-norm of both parallel and perpendicular components of the random walk for each application of P_iM. Let $z_0 \triangleq \varphi$ and $z_i = z_{i-1}P_iM$ for $i \in [t]$. By triangle inequality and Lemma 5 and 6, for every $i \in [t]$,

$$\begin{aligned}\|z_i^{\parallel}\|_\pi = \|(z_{i-1}P_iM)^{\parallel}\|_\pi = \|((z_{i-1}^{\parallel} + z_{i-1}^{\perp})P_iM)^{\parallel}\|_\pi &\leq \|(z_{i-1}^{\parallel}P_iM)^{\parallel}\|_\pi + \|(z_{i-1}^{\perp}P_iM)^{\parallel}\|_\pi \\ &\leq (1+(e^r-1)\mu)\,\|z_{i-1}^{\parallel}\|_\pi + (2r\sqrt{\mu})\,\|z_{i-1}^{\perp}\|_\pi,\end{aligned}$$

and similarly,

$$\begin{aligned}\|z_i^{\perp}\|_\pi \leq \|(z_{i-1}^{\parallel}P_iM)^{\perp}\|_\pi + \|(z_{i-1}^{\perp}P_iM)^{\perp}\|_\pi &\leq (2r\lambda\sqrt{\mu})\,\|z_{i-1}^{\parallel}\|_\pi + (e^r\lambda)\,\|z_{i-1}^{\perp}\|_\pi \\ &\leq (2r\lambda\sqrt{\mu})\,\|z_{i-1}^{\parallel}\|_\pi + \sqrt{\lambda}\|z_{i-1}^{\perp}\|_\pi,\end{aligned}$$

where the last inequality holds when $r \leq (1/2)\log(1/\lambda)$ i.e. $e^r \leq 1/\sqrt{\lambda}$. The reason to require $r \leq (1/2)\log(1/\lambda)$ is that we can guarantee the perpendicular component is *shrinking* (by a factor of $\sqrt{\lambda} < 1$) after each step.

Now let $\alpha_0 = \|z_0^{\parallel}\|_\pi = 1$ and $\beta_0 = \|z_0^{\perp}\|_\pi$, and define for $i \in [t]$,

$$\alpha_i = (1+(e^r-1)\mu)\,\alpha_{i-1} + (2r\sqrt{\mu})\,\beta_{i-1} \quad \text{and} \quad \beta_i = (2r\lambda\sqrt{\mu})\,\alpha_{i-1} + \sqrt{\lambda}\beta_{i-1}.$$

One can prove by induction easily that $\|z_i^{\parallel}\|_\pi \leq \alpha_i$ and $\|z_i^{\perp}\|_\pi \leq \beta_i$ for every $i \in [t]$, and α_i's are strictly increasing. Therefore, bounding the moment generating function $\mathrm{E}[e^{rX}] = \|z_t^{\parallel}\|_\pi \leq \alpha_t$ boils down to bounding the recurrence relation for α_i and β_i.

Observe that in the recurrence relation, only the coefficient $(1+(e^r-1)\mu) > 1$ while the remaining coefficients $(2r\sqrt{\mu}), (2r\lambda\sqrt{\mu})$, and $\sqrt{\lambda}$ are all less than 1 if r is chosen sufficiently small. This suggests, intuitively, α_i's terms will eventually dominate. This provides us a guide to reduce the recurrence relation to a single variable. In particular, one can show that $\beta_i \leq 2r\left(\sum_{j=0}^{i-1}\sqrt{\lambda^{j+2}\mu}\right)\alpha_{i-1} + \sqrt{\lambda^i}\beta_0$ for every $i \in [t]$, by expanding the recurrence relation and using the fact that α_i's are increasing. Also, by substituting β_{i-1}, we get $\alpha_1 \leq (1+(e^r-1)\mu) + 2r\sqrt{\mu}\beta_0$ and $\alpha_i \leq \left(1+(e^r-1)\mu + 4r^2\sqrt{\mu}\left(\sum_{j=0}^{i-2}\sqrt{\lambda^{j+2}\mu}\right)\right)\alpha_{i-1} + 2r\sqrt{\lambda^{i-1}\mu}\beta_0$ for every $2 \leq i \leq t$. One can then show that

$$\alpha_t \leq \left(1+\frac{8r\sqrt{\mu}\beta_0}{1-\lambda}\right)(1+(e^r-1))\prod_{i=2}^{t}\left(1+(e^r-1)\mu + 4r^2\sqrt{\mu}\left(\sum_{j=0}^{i-2}\sqrt{\lambda^{j+2}\mu}\right)\right)$$

which can be further shown to be bounded by

$$2\max\left\{1, \frac{8r\sqrt{\mu}}{1-\lambda}\right\}\|\varphi\|_\pi \exp\left\{\left((e^r-1)+\frac{8r^2}{(1-\lambda)}\right)\mu t\right\}.$$

through elementary analysis. Recall that our goal is to choose an r to bound $\mathrm{E}[e^{rX}]/e^{r(1+\delta)\mu t}$. Choosing $r = \min\{1/2, \log(1/\lambda)/2, 1-\sqrt{\lambda}, (1-\lambda)\delta/18\} = (1-\lambda)\delta/18$, we complete the proof of Claim 3.2.

◀

Before completing this subsection, we make a final remark. Our proof also works even for the case $\mathrm{E}_\pi[f_i(v)]$ are *different* for different values of i, which results in a more general Chernoff type bound based on spectral expansions. This more general result, as far as we know, has not been noted in existing literatures with the exception of Healy [7], who gave a Chernoff bound of this kind with stronger assumptions for regular graphs, although the analysis given by Lezaud [12] or Wagner [16] also appears to be generalizable as well. On the other hand, this strengthened result of Claim 3.2 does not seem to be sufficient to remove the requirement that $\mathrm{E}_\pi[f_i(v)]$ are the same for Theorem 3.

3.3 Continuous Time Case

We now generalize our main result to cover the continuous time chains. The analysis is similar to the one presented by Lezaud [12] and is presented in the full version of this paper.

▶ **Theorem 7.** *Let Λ be the generator of an ergodic continuous time Markov chain with state space $[n]$ and mixing time $T = T(\epsilon)$. Let $\{v_t : t \in \mathbf{R}^+\}$ be a random walk on the chain starting from an initial distribution φ such that v_t represents the state where the walk stay at time t. Let $\{f_t : [n] \to [0,1] \mid t \in \mathbf{R}^+\}$ be a family of functions such that $\mu = \mathrm{E}_{v \leftarrow \pi}[f_t(v)]$ for all t. Define the weight over the walk $\{v_s : s \in \mathbf{R}^+\}$ up to time t by $X_t \triangleq \int_0^t f_s(v_s)ds$. There exists a constant c such that*

$$1.\ \Pr[X \geq (1+\delta)\mu t] \leq \begin{cases} c\|\varphi\|_\pi \exp\left(-\delta^2\mu t/(72T)\right) & \text{for } 0 \leq \delta \leq 1 \\ c\|\varphi\|_\pi \exp\left(-\delta\mu t/(72T)\right) & \text{for } \delta > 1 \end{cases}$$

$$2.\ \Pr[X \leq (1-\delta)\mu t] \leq c\|\varphi\|_\pi \exp\left(-\delta^2\mu t/(72T)\right) \quad \text{for } 0 \leq \delta \leq 1$$

References

1 D. Aldous. Some inequalities for reversible markov chains. *Journal of London Mathematical Society*, 25:564–576, 1982.

2 K. Avrachenkov, B. Ribeiro, and D. Towsley. Improving random walk estimation accuracy with uniform restart. In *7th Workshop on Algorithms and Models for the Web Graphs (WAW 2010)*, 2010.

3 A. Fill. Eigenvalue bounds on convergence to stationarity for nonreversible markov chains, with an application to the exclusion process. *Annals of Applied Probability*, 1, Number 1:62–87, 1991.

4 A. Frieze, P. Melsted, and M. Mitzenmacher. An analysis of random-walk cuckoo hashing. *SIAM Journal of Computing*, 2011.

5 D. Gillman. A chernoff bound for random walks on expander graphs. *SIAM Journal on Computing*, 27(4), 1997.

6 O. Goldreich and D. Ron. On testing expansion in bounded degree graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 2000.

7 A. Healy. Randomness efficient sampling within nc^1. *Computational Complexity*, 17(1), 2008.

8 W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30, 1963.

9 M. Jerrum and A. Sinclair. *The Markov chain Monte Carlo method: an approach to approximate counting and integration.* PWS Publishing, Boston, MA, USA, 1996.

10 N. Kahale. Large deviation bounds for markov chains. *Combinatorics, Probability, and Computing*, 6(4), 1997.

11 C. A. Leon and F. Perron. Optimal hoeffding bounds for discrete reversible markov chains. *Annals of Applied Probability*, 14(2), 2004.

12 P. Lezaud. Chernoff-type bound for finite markov chains. *Annals of Applied Probability*, 8(3):849–867, 1998.

13 A Mohaisen, A. Yun, and Y Kim. Measuring the mixing time of social graphs. In *IMC'10 Proceedings of the 10th Annual Conference on Internet Measurement*, 2010.

14 A. Sinclair. Improved bounds for mixing rates of markov chains and multicommodity flow. *Combinatorics, Probability, and Computing*, 1:351–370, 1992.

15 C. Tekin and M. Liu. Online algorithms for the multi-armed bandit problem with markovian rewards. In *48th Annal Allerton Conference on Communication, Control, and Computing (Allerton)*, 2010.

16 R. Wagner. Tail estimates for sums of variables sampled by a random walk. *Combinatorics, Probability, and Computing*, 17(2), 2008.
17 A. Wigderson and D. Xiao. A randomness-efficient sampler for matrix valued functions and applications. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2005.

A Construction of Mixing Markov Chain with No Spectral Expansion

In this section, we show that any ergodic Markov chain M with mixing time $T = T(1/4)$ can be modified to a chain M' such that M' has mixing time $O(T)$ but spectral expansion $\lambda(M') = 1$.

Our modification is based on the following simple observation. Let M' be an ergodic Markov chain with stationary distribution π'. If there exist two states v and v' such that (i) $M'_{v,v'} = 1$, i.e., state v leaves to state v' with probability 1, and (ii) $M'_{u,v'} = 0$ for all $u \neq v$, i.e., the only state transits to v' is v, then $\lambda(M') = 1$: Note that in this case, $\pi'(v) = \pi'(v')$ since all probability mass from v leaves to v', which receives probability mass only from v. Consider a distribution x whose probability mass all concentrates at v, i.e., $x_v = 1$ and $x_u = 0$ for all $u \neq v$. One step walk from x results in the distribution xM' whose probability mass all concentrates at v'. By definition, $\|x\|_{\pi'} = \|xM'\|_{\pi'}$ and thus $\lambda(M') = 1$.

Now, let M be an ergodic Markov chain with mixing time $T = T(1/4)$ and stationary distribution π. We shall modify M to a Markov chain M' that preserves the mixing-time and satisfies the above property. We mention that it is not hard to modify M to satisfy the above property. The challenge is to do so while preserving the mixing-time. Our construction is as follows.

- For every state v in M, we "split" it into three states $(v, in), (v, mid), (v, out)$ in M'.
- For every state (v, in) in M', we set $M'_{(v,in),(v,in)} = M'_{(v,in),(v,mid)} = 1/2$, i.e., (v, in) stays in the same state with probability $1/2$ and transits to (v, mid) with probability $1/2$.
- For every state (v, mid) in M', we set $M'_{(v,mid),(v,out)} = 1$, i.e., (v, mid) always leaves to (v, out).
- For every pairs of states u, v in M, we set the transition probability $M'_{(u,out),(v,in)}$ from (u, out) to (v, in) to be $M_{u,v}$.

It is not hard to verify that the modified chain M' is well-defined, ergodic, and satisfies the aforementioned property (namely, (v, mid) leaves to (v, out) with probability 1 and is the only state that transits to (v, out)). It remains to show that M' has mixing-time $O(T)$. Toward this goal, let us define yet another Markov chain C that consists of three states $\{in, mid, out\}$ with transition probability $C_{in,in} = C_{in,mid} = 1/2$, and $C_{mid,out} = C_{out,in} = 1$. Clearly, C is ergodic and has constant mixing-time. Now, the key observation is that a random walk on M' can be decomposed into walks on M and C in the following sense: every step on M' corresponding to a step on C in a natural way, and one step on M' from (u, out) to (v, in) can be identified as a step from u to v in M. Note that the walks on M and C are independent, and in expectation, every 4 steps of walk on M' induce one step of walk on M. It is not hard to see from these observation that the mixing time of M' is at most $8T$.

Compressed Membership for NFA (DFA) with Compressed Labels is in NP (P) *

Artur Jeż[1]

1 **University of Wrocław**
ul. Joliot-Curie 15, 50-383 Wrocław, Poland
aje@cs.uni.wroc.pl

Abstract

In this paper, a compressed membership problem for finite automata, both deterministic (DFAs) and non-deterministic (NFAs), with compressed transition labels is studied. The compression is represented by straight-line programs (SLPs), i.e. context-free grammars generating exactly one string. A novel technique of dealing with SLPs is introduced: the SLPs are recompressed, so that substrings of the input text are encoded in SLPs labelling the transitions of the NFA (DFA) in the same way, as in the SLP representing the input text. To this end, the SLPs are *locally decompressed* and then *recompressed* in a uniform way. Furthermore, in order to reflect the recompression in the NFA, we need to modify it only a little, in particular its size stays polynomial in the input size.

Using this technique it is shown that the compressed membership for NFA with compressed labels is in NP, thus confirming the conjecture of Plandowski and Rytter [21] and extending the partial result of Lohrey and Mathissen [14]; as this problem is known to be NP-hard, we settle its exact computational complexity. Moreover, the same technique applied to the compressed membership for DFA with compressed labels yields that this problem is in P, and this problem is known to be P-hard.

1998 ACM Subject Classification F.4.3 Formal Languages, F.4.2 Grammars and Other Rewriting Systems, F.2.2 Nonnumerical Algorithms and Problems, F.1.1 Models of Computation

Keywords and phrases Compressed membership problem, SLP, Finite Automata, Algorithms for compressed data

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.136

1 Introduction

1.1 Compression and Straight-Line Programms

Due to ever-increasing amount of data, compression methods are widely applied in order to decrease the data's size. The stored data is processed from time to time and decompressing it on each occasion is wasteful. Thus there is a large demand for algorithms working directly on the compressed representation of the data, without explicit decompression. Such task is not as desperate, as it may seem: it is a popular outlook that compression basically extracts the hidden structure of the text and if the compression rate is high, the text must have a lot of internal structure. So if data is compressed well, it has a structure that can be exploited by algorithms. Indeed, efficient algorithms for fundamental text operations (pattern matching,

* This work was partially supported by NCN grant number DEC-2011/01/D/ST6/07164, 2011–2014 and by a personal scholarship funded by the Human Capital Programme.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 136–147

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

checking equality, etc.) are known for various practically used compression methods (LZ, LZW, etc.) [2, 3, 5].

Practical compression methods, like LZW or LZ variants, differ and so only algorithms for data compressed using them are also different. This leads to a plethora of algorithms for various compression variants and string operations. However, a different approach is also explored: for some applications and for most of theory-oriented considerations it would be useful to *model* the practical compression standard by a more mathematically well-founded method. This idea, lay at the foundations of the notion of *Straight-Line Programms* (SLP), whose instance can be simply seen as context-free grammars generating exactly one string.

SLPs are the most popular theoretical model of compression. This is on one hand motivated by a simple, 'clean' and appealing definition, on the other hand, they model the LZ compression standard: each LZ compressed text can be converted into an equivalent SLP with only logarithmic, with respect to decompressed data's size, overhead (and in polynomial time) while each SLP can be converted to an equivalent SLP with just a constant overhead (and in polynomial time).

The approach of modelling compression by SLP in order to develop efficient algorithms turned out to be fruitful. Algorithmic problems for SLP-compressed input strings were considered and successfully solved [10, 11, 18]. The recent state-of-the-art efficient algorithms for pattern matching in LZ and LZW compressed text essentially use the reformulation of LZW and LZ methods in terms of SLPs [2, 3]. SLPs found their usage also in programme verification [4, 9]. Surprisingly, while SLPs were introduced mainly as a model for practical applications, they turned out to be useful also in strictly theoretical branches of computer science, for instance, in the word equations [20, 19]; in particular, the currently best PSPACE bound was obtained in this fashion by Plandowski [19].

1.2 Membership problem

As SLPs are used both in theoretical and applied research in computer science, tools for dealing with them should be developed. In particular, one should be aware that whenever working with strings, these may be supplied as respective SLPs. Hence, all the usual string problems should be reinvestigated in the compressed setting, as the classical algorithms may not apply directly, be inefficient or the problems themselves may become computationally difficult.

From language theory point of view, the crucial questions stated in terms of strings, is the one of compressed string recognition. To be more precise, we consider classic membership problems, i.e. recognition by automata, generation by a grammar etc., in which the input is supplied as an SLP. We refer to such problems as *compressed membership problems*. These were first studied in the pioneering work of Plandowski and Rytter [21], who considered compressed membership problem for various formalism for defining languages. Already in this work it was observed that we should precisely specify, what part of the input is compressed. Clearly the input string, but what about the language representation (i.e. regular expression, automaton, grammar, etc.). Should it be also compressed or not? Both variant of the problem are usually considered, with the following naming convention: when only the input string is compressed, we use a name *compressed membership*, when also the language representation, we prepend *fully* to the name.

In years to come, the compressed membership problem was investigated for various language classes [5, 7, 12, 13, 21]. Compressed word problem for groups and monoids [12, 15, 16], which can be seen as a generalisation of membership problem, was also investigated.

Despite the large attention in the research community, the exact computational complexity

of some problems remained open. The most notorious of those is the fully compressed membership problem (FCMP) for NFA, considered already in the work of Plandowski and Rytter [21]. Here, the compression of NFA is done by allowing it to have transitions by strings, instead of single letters, and representing these strings as SLPs.

It is relatively easy to observe that the compressed membership problem for the NFA is in P, however, the status of the fully compressed variant remained open for a long time. Some partial results were obtained by Plandowski and Rytter [21], who observed that it is in PSPACE and is NP-hard for the case of one-letter alphabet, both of these bounds being relatively simple. Moreover, they showed that this problem is in NP for some particular cases, for instance, for one-letter alphabet. Further work on the problem was done by Lohrey and Mathissen [14], who demonstrated that if the strings defined by SLP have polynomial periods, the problem is in NP, and when all strings are highly aperiodic, it is in P. Concerning the case of DFAs, it is known that even for a fixed regular language, the compressed membership is P-hard [17, 1], and no upper-bound better than PSPACE was known.

1.3 Our results and techniques

We establish the computational complexity of fully compressed membership problems for both NFAs and DFAs.

▶ **Theorem 1.** *Fully compressed membership problem for NFA is in* NP*, for DFA it is in* P*.*

Our approach to the problem is essentially different than the ones of Plandowski and Rytter [21] and Lohrey and Mathissen [14]. The earlier work focused on the properties of strings described by SLPs. We take a completely different route: we analyse and change the way strings are described by the SLPs in instance. That is, we focus on the SLPs, and not on the encoded strings. Roughly, our algorithm aims at having all the strings in the instance compressed 'in the same way'. To achieve this goal, we decompress the SLPs. Since the compressed text can be exponentially long, we do this locally: we introduce explicit strings into the rules' bodies. Then, we recompress these explicit strings uniformly. Since such pieces of text are compressed in the same way, we can 'forget' about the original substrings of the input and treat the introduced nonterminals as atomic letters. Such recompression shortens the text significantly: one 'round' of recompression, in which every pair of letters that was present at the beginning of the 'round' is compressed, should shorten the encoded strings by a constant factor.

Other application of the technique

We stress that the idea of local decompression and recompression of SLP is new and promising: there is hope that it can be applied to other problem related to SLPs. In fact between the submission of this work and its acceptance the author successfully applied this technique in fully compressed pattern matching [6], obtaining a faster algorithm for this problem.

2 Preliminaries

2.1 Straight line programmes

Formally, a *Straight line programme* (SLP) is context free grammar G with a language consisting of exactly one string. Usually it is assumed that G is in a *Chomsky normal form*, i.e. each production is either of the form $X \to YZ$ or $X \to a$. By this assumption, strings

defined by G's nonterminals have length at most 2^n; since our algorithm will replace some substrings by shorter ones, none string defined by SLPs during the run of algorithm will exceed this length.

We denote the string defined by nonterminal A by $\mathrm{val}(A)$ (like *value*). A *symbol* is either a letter or a nonterminal. The notion of val extends to strings of symbols in an obvious way.

2.2 Input

The instance of the fully compressed membership problem (FCMP) for NFA consists of an input string, represented by an SLP, and an NFA N, whose transitions may be labelled by SLPs.

For our purposes it is more convenient to assume, that all SLPs are given as a single context free grammar G with a set of nonterminals $\mathcal{X} = \{X_1, \ldots, X_n\}$, the input string is defined by X_n and the NFAs transitions are labelled with nonterminals of G. While we require that the input grammar is in the Chomsky normal form, during the algorithm we allow the grammar to be in a slightly more general form, described by the following conditions:

$$\text{each nonterminal has exactly one production, which is of the form} \tag{1a}$$

$$X_i \to uX_jvX_k \quad \text{or} \quad X_i \to uX_jv \quad \text{or} \quad X_i \to u, \quad \text{where } u, v \in \Sigma^* \text{ and } j, k < i, \tag{1b}$$

$$\text{if } \mathrm{val}(X_i) = \epsilon \text{ then } X_i \text{ does not appear in the rules' bodies.} \tag{1c}$$

The strings u, v and their substrings appear *explicitly* in a rule, this notion is introduced to distinguish them from the substrings of $\mathrm{val}(X_i)$.

Without loss of generality we may assume that the input string starts and ends with designated, unique symbols, denoted as \$ and #. These are not essential, however, the first and last letter of $\mathrm{val}(X_n)$ need to be treated in a somewhat special manner, furthermore, this applies to these letters' appearances in the NFA as well. Having special symbols for the first and last letter makes the analysis smoother.

2.3 Input size, complexity classes

The size $|G|$ of the representation of grammar G is the sum of length of G's rules' bodies. The size $|N|$ of the representation of NFA N is the sum of number of its states and transitions. The size $|\Sigma|$ of alphabet Σ is simply the number of elements in Σ.

The input (or, in general, current instance) size is polynomial in N, G, Σ and n, which denotes the number of nonterminals in G. One of the crucial properties of our algorithm is that n only decreases during the run of the algorithm.

By npolytime (polytime) we denote the class of algorithms running in non-deterministic (deterministic, respectively) polynomial time, and by NP (P, respectively) the corresponding complexity classes of the decision problems.

2.4 Automata, paths and labels, determinism

Since we investigate automata, proofs deal mainly with (accepting) paths for strings. the constructed NFAs have transitions labelled with either letters, or non-terminals of G. That is $\delta \subseteq Q \times (\Sigma \cup \mathcal{X}) \times Q$. Consequently, a *path* $\mathcal{P}$ from state p_1 to p_{k+1} is a sequence $\alpha_1\alpha_2 \ldots \alpha_k$, where $\alpha_i \in \Sigma \cup \mathcal{X}$ and $\delta(p_i, \alpha_i, p_{i+1})$. We write that $\mathcal{P}$ *induces* such a list of labels. The $\mathrm{val}(\mathcal{P})$ *defined* by such a path $\mathcal{P}$ is simply $\mathrm{val}(\alpha_1 \ldots \alpha_k)$. We also say that $\mathcal{P}$ is a path for a

string val($\mathcal{P}$). A path is *accepting*, if it ends in an accepting state. A string w is accepted by N if there is an accepting path from the starting state for w.

We consider also DFAs with compressed labels. Let us comment, what 'determinism' means here: a NFA with compressed labels is a *deterministic*, when for each state q and any two transitions from q labelled with α and α', the first letters of val(α) and val(α') are different. One could define determinism meaningfully in other ways, however, all these notions seem to be polynomially equivalent.

2.5 Known results

We use the following basic result, which states that the FCMP, when the input string is over a one-letter alphabet, is in NP for NFA and in P for DFA.

▶ **Lemma 2** (cf. [21, Theorem 5]). *The FCMP restricted to the input string over an alphabet $\Sigma = \{a\}$ is in* NP *for NFA and in* P *for DFA.*

The second claim is trivial and the first claim can be easily inferred from the result of Plandowski and Rytter [21, Theorem 5], who proved that this problem is in NP, when $\Sigma = \{a\}$, i.e. also transitions in the NFA are labelled by powers of a only. In such a case a path in NFA exists if and only if it satisfies an Eulerian-type condition: each state is entered and leaved the same number of times. Since each edge is used at most expontentially many times, a description of such a path can be guessed and then it can be verified whether it satisfies the condition and defines a word of appropriate length.

3 Basic classifications and outline of the algorithm

In this section we present the outline of the algorithm for FCMP for NFAs. Its main part consist of recompression, i.e. replacing strings appearing in val(X_n) by shorter ones. In some cases, such replacing is harder, in other easier. It should be intuitively clear that it depends on the position of letters inside encoded strings: if a is a first or last letter of some val(X_i), then recompressing strings including a looks difficult, as such strings can be split into different nonterminals and recompression requires heavy modification of G, or even rebuilding of the NFA. On the other hand, if a letter a is only 'inside' strings encoded by nonterminals, its compression is done only 'inside' rules of G, which seems easy. Thus, before we state the algorithm, we firstly introduce classification of letters (and strings) into 'easy' and 'difficult' to compress.

3.1 Crossing appearances, types of letters, maximal blocks

We say that a string w has a *crossing appearance in a (string defined by) nonterminal* X_i with a production $X_i \to uX_jvX_k$, if w appears in val(X_i), but this appearance is not contained in neither u, v, val(X_j) nor val(X_k). Intuitively, this appearance 'crosses' the symbols in $u\,\mathrm{val}(X_j)v\,\mathrm{val}(X_k)$, i.e. at the same time part of w is in the explicit substring (u or v) and part is in the compressed strings (val(X_j) or val(X_k)). This notion is similarly defined for nonterminals with productions of the form $X_i \to uX_jv$, productions of the form $X_i \to u$ clearly do not have crossing appearances.

A string w has a *crossing appearance in the NFA* N, if there is a path in N inducing list of labels $\alpha_1\alpha_2$, where $\alpha_1, \alpha_2 \in \mathcal{X} \cup \Sigma$ with at least one of α_1, α_2 being a nonterminal, such that w appears in a val($\alpha_1\alpha_2$), but this appearance is not contained in the val(α_1), nor in val(α_2). The intuition is similar as in the case of crossing appearance in a rule: it is possible

that a string w is split between two transitions' labels. Still, there is nothing difficult in consecutive letter transitions, thus we treat such a case as a simple one.

We say that a pair of different letters ab is a *crossing pair*, if ab has a crossing appearance of any kind. Otherwise, such a pair is *non-crossing*.

We say that a letter $a \in \Sigma$ is *left-outer* (*right-outer*), if there is a nonterminal X_i, such that a is the leftmost (rightmost, respectively) symbol in $\mathrm{val}(X_i)$. A letter is *outer*, if it is left-outer or right-outer. Otherwise, the letter is *inner*. Notice that if a pair ab is crossing, then b is left-outer or a is right-outer. The outer letters and crossing pairs correspond to the intuitive notion of being 'hard' to compress.

The following lemma shows that while G may encode long strings, they have relatively few different short substrings and few outer letters.

▶ **Lemma 3.** *There are at most $2n$ different outer letters and at most $|G| + 3n$ different pairs of letters appearing in $\mathrm{val}(X_1), \ldots, \mathrm{val}(X_n)$.*

The set of outer letters, the set of crossing pairs and the set of non-crossing pairs appearing in $\mathrm{val}(X_1), \ldots, \mathrm{val}(X_n)$ can be computed in polytime.

The notions of (non-) crossing pairs do not apply to aa, still, an analog can be defined: for a letter $a \in \Sigma$ we say that a^ℓ is a a's *maximal block of length* ℓ (or simply ℓ-*block*), if it appears in some string defined by some nonterminal and it is surrounded by letters other than a, formally, if there exist two letters $x, y \in \Sigma$, where $x \neq a \neq y$ and a nonterminal X_i, such that $xa^\ell y$ is a substring of $\mathrm{val}(X_i)$. Similarly to crossing pairs, it can be shown that there are not too many different maximal blocks of a.

▶ **Lemma 4.** *For a letter a there are at most $|G|$ different lengths of a's maximal blocks in $\mathrm{val}(X_1), \ldots, \mathrm{val}(X_n)$. The set of these lengths can be calculated in* polytime.

3.2 Outline of the algorithm

Our algorithm consists is based on two main operations performed on strings encoded by G

blocks compression of a For each a^ℓ that is and ℓ-block in $\mathrm{val}(X_n)$ and $\ell > 0$, replace all a's ℓ-blocks in $\mathrm{val}(X_1), \ldots, \mathrm{val}(X_n)$ by a fresh letter a_ℓ. Modify N accordingly.

pair compression of ab For two *different* letters ab replace each of ab in $\mathrm{val}(X_1), \ldots, \mathrm{val}(X_n)$ by a fresh letter c. Modify N accordingly.

We denote the string obtained from w by a's blocks compression by $BC_a(w)$, and the string obtained by compression of a pair ab into c by $PC_{ab\to c}(w)$.

We adopt the following notational convention throughout rest of the paper: whenever we refer to a letter a_ℓ, it means that the last block compression was done for a and a_ℓ replaced a's ℓ-blocks.

The main idea behind the algorithm is that block compression and pair compression shorten the encoded texts significantly. The general schema is given in Algorithm 1.

There are two important remarks to be made:

- there is no explicit non-deterministic operation in the code, however, it appears implicitly in the term 'modify the NFA accordingly' in lines 4 and 10. Roughly, to perform such a modification, one needs to solve FCMP for string a^ℓ, and this is known to be NP-hard.
- the compression (both of pairs and blocks) is never applied to \$, nor to #. The markers were introduced so that we do not bother with strange behaviour when first or last letter is compressed, and so we do not touch the markers.

Ideally, each letter of the input is compressed and so the $|\mathrm{val}(X_n)|$ halves in each iteration of the main loop. The worst case scenario is not far from the ideal behabviour.

Algorithm 1 Outline of the CompMem, which tests compressed membership

```
1: while |val(X_n) > n| do
2:     while something changed do
3:         for a: inner letter do
4:             compress blocks of a, modify N accordingly
5:         for non-crossing pair ab in val(X_n), a, b ∉ {$, #} do
6:             compress ab, modify N accordingly
7:     L ← list of outer letters, except $ and #
8:                                  ▷ Including letters introduced in line 4 and 6
9:     for a ∈ L do
10:        compress blocks of a, modify N accordingly
11:        for each a_ℓb in val(X_n) do
12:            compress a_ℓb, modify N accordingly
13: Decompress X_n and solve the problem naively.
```

▶ **Lemma 5.** *There are $\mathcal{O}(n)$ executions of the loop in line 1 of* CompMem.

▶ Remark. Notice that pair compression $PC_{ab\to b}$ is in fact introducing a new nonterminal with a production $c \to ab$, similarly BC_a. Hence, CompMem creates new SLPs, that encode strings from the instance. However, these new nonterminals are never expanded, they are always treated as individual symbols. Thus it is better to think of them as letters. Moreover, the analysis of running time of CompMem relies on the fact that no new nonterminals are introduced by CompMem.

4 Details

In this section we describe in detail how to implement the block compression and pair compression and how to modify the NFA. In particular, we are going to formulate the connections between NFA and SLPs preserved during CompMem.

4.1 Invariants

The invariants below describe the grammar kept by CompMem.

SLP 1 The set of used nonterminals is a subset of $\mathcal{X} = \{X_1, \ldots, X_n\}$ and the productions are of the form described in (1).

SLP 2 The nonterminal X_n has a production $X_n \to \$uX_{n-1}v\#$, where $u, v \in (\Sigma \setminus \{\$, \#\})^*$; \$, # are not used in other productions.

The following invariants represent the constraints on the NFA.

Aut 1 every transition of N is labelled by a single letter of Σ (*letter transition*) or by a nonterminal (*nonterminal transition*) that does not define ϵ, each nonterminal labels at most one transition. No transition is labelled with X_n.

Aut 2 there is a unique starting state that has a unique outgoing transition labelled by letter \$, and no incoming transitions; there is no other transition by \$. Similarly, there is a unique accepting state that has a unique incoming transition labelled by letter #, it does not have any outgoing transitions; there is no other transition by # in N.

CompMem will preserve (SLP 1)–(Aut 2), and we shall always assume that the input of the subroutines satisfies (SLP 1)–(Aut 2).

We assume that the input instance satisfies (SLP 1)–(Aut 2), moreover that the input grammar is in the Chomsky normal form. It is routine to transform (in polytime) the input instances not satisfying these conditions into equivalent instances that satisfy them.

4.2 Compression of non-crossing pairs and inner letters

The compression of non-crossing pairs and block compression for inner letters is intuitively easy: whenever these appear in strings encoded by G or on paths in N, they cannot be split between nonterminals or between transitions. So we replace their explicit appearances in the grammar and in the NFA. This is formalised and shown in this subsection

Consider a non-crossing pair ab. Since it is non-crossing, it can only appear in the the explicit strings in the rules of G. Hence, compressing ab into a fresh letter c consists of replacing each explicit ab by c in rules' bodies. Still, ab can appear on a path in N. But since ab is non-crossing, this can be either wholly inside a nonterminal transition (and so compression was already taken care of), or on two consecutive letter transitions. This is also easy to handle: whenever there is a path from p to q by a string ab, we introduce a new letter transition by c from p to q. This description is formalised in PairComp.

To distinguish between the input and output G and N, we utilise the following convention: 'unprimed' names refer to the input (like G, X_i, N), while 'primed' symbols refer to the output (like G', X'_i, N'). This convention is used in lemmata concerning algorithms through the paper.

Algorithm 2 PairComp(ab, c), which compresses a non-crossing pair ab into c

```
for each production X_i → α do
    replace each explicit ab in α by c
for states p, q do
    if δ_N(p, ab, q) then put a transition δ_N(p, c, q)
```

▶ **Lemma 6.** PairComp *runs in* polytime *and preserves (SLP 1)–(Aut 2). When applied to a non-crossing pair of letters ab, where $a, b \notin \{\$, \#\}$, it implements the pair compression, i.e.* $\mathrm{val}(X'_i) = PC_{ab\to c}(\mathrm{val}(X_i))$*, for each* X_i*.*

N' recognises $\mathrm{val}(X'_n)$ *if and only if N recognises* $\mathrm{val}(X_n)$*. If N is a DFA, so is N'.*

We can apply the same approach to the inner letters block compression. However, in this case, the modification of N uses non-determinism.

Since a is an inner letter, it cannot appear as the last or first letter of any nonterminal, and so every maximal block of a in $\mathrm{val}(X_1), \ldots, \mathrm{val}(X_n)$ is an explicit substring in one of the rule' bodies; so we simply replace explicit a^ℓ by a fresh letter a_ℓ in rules' bodies. Before considering the NFA, notice that as a is an inner letter, a^ℓ cannot have a crossing appearance in N, and no nonterminal defines a^ℓ. Hence, when a^ℓ is a substring of a string defined by a path in N, then a^ℓ appears wholly inside a nonterminal transition, or a^ℓ labels a path using letter transitions only. The former case is taken care of by compression of a maximal blocks in G, and in the latter case for each a^ℓ and each pair of states P and q we check whether there is a path for a^ℓ from p to q using letter transitions only.

Algorithm 3 BlockComp(a), which compresses inner letter a blocks

1: establish the lengths $\ell_1, \ldots, \ell_k$ of a's maximal block
2: **for** each a^{ℓ_m} **do**
3: **for** each production $X_i \to \alpha$ **do**
4: replace every explicit maximal block a^{ℓ_m} in α by a_{ℓ_m}
5: **for** states p, q in N **do**
6: **if** $\delta_N(p, a^{\ell_m}, q)$ **then** ▷ Check non-deterministically, see Lemma 2
7: put a transition $\delta_N(p, a_{\ell_m}, q)$

▶ **Lemma 7.** *Suppose that* BlockComp *is applied for inner letter $a \notin \{\$, \#\}$. Then it preserves (SLP 1)–(Aut 2) and properly implements maximal block compression, i.e.* $\mathrm{val}(X_i') = BC_a(\mathrm{val}(X_i))$ *for each X_i.*

The operations in line 6 of BlockComp *can be performed in* npolytime*, other operations can be performed in* polytime*.*

Each of the new letters a_ℓ is inner. N recognises $\mathrm{val}(X_n)$ *if and only if N' recognises* $\mathrm{val}(X_n')$ *for some non-deterministic choices. If N is DFA, so is N'.*

4.3 Compression of outer letters and crossing pairs

Now, we turn our attention to the compression of outer letters and crossing pairs. The outline is as follows: we fix an outer letter a and modify the instance, so that a becomes inner. Then, BlockComp is applied to a. Next, we want to compress each pair of the form $a_\ell b$. Such a pair can be crossing, as b can be a left-outer letter. Thus, we modify the instance again, so that none of $a_\ell b$ is a crossing pair so afterwards we compress it using PairComp.

4.3.1 Transforming an outer letter to an inner letter

The reason, why a is an outer letter, is that it is the first or the last symbol in some $\mathrm{val}(X_i)$. To make it an inner letter, it is enough to remove each nonterminal's a-prefix and a-suffix. To be more precise: fix i and let $\mathrm{val}(X_i) = a^{\ell_i} u a^{r_i}$, where u does not start nor end with a. Then our goal is to modify G so that $\mathrm{val}(X_i') = u$. (If $\mathrm{val}(X_i)$ is a power of a, we simply give $u = \epsilon$ and $r_i = 0$.) This can be done in a bottom-up fashion, starting from X_1: it is enough to calculate and memorise the lengths of the a-prefixes and a-suffixes for consecutive nonterminals, see the operations in lines 1–4 of OutToIn. Then we need to modify the NFA accordingly: it is enough to replace the transition labelled with X_i by path consisting of three transitions, labelled with a^{ℓ_i}, X_i' and a^{r_i}.

The removed a-prefixes and a-suffixes can be exponentially long, and so we store them in the rules in a succinct way, i.e. a^ℓ is represented as (a, ℓ); the size of representation of ℓ is $\mathcal{O}(\log \ell)$, that is, linear in n. We say that such a grammar is in an *a-succinct* form. The situation is similar for the NFA, as it might have transitions labelled with a^ℓ, which are stored in succinct way as well. We say that N satisfies *a-relaxed* (Aut 1), if its transitions are labelled by nonterminals, a single letter or by a^ℓ, where $\ell \leq 2^n$.

▶ **Lemma 8.** OutToIn(a) *for $a \notin \{\$, \#\}$ runs in* polytime *time and preserves (SLP 1)–(Aut 2), except that it a-relaxes (Aut 1). G' is in the a-succinct form.*

Let $\mathrm{val}(X_i) = a^{\ell_i} u_i a^{r_i}$*, where u_i does not begin, nor end with a. After* OutToIn $\mathrm{val}(X_i') = u_i$*. In particular the letter a is inner.*

N accepts $\mathrm{val}(X_n)$ *if and only if N' accepts* $\mathrm{val}(X_n')$*. If N is a DFA, so is N'.*

Algorithm 4 OutToIn(a), which changes an outer letter a to an inner letter.

for $i = 1 \dots n$ **do**
  let the production for X_i be $X_i \to \alpha_i$
  let a^{ℓ_i}, a^{r_i} be the the explicit a-prefix and a-suffix of α_i
  remove prefix a^{ℓ_i} and suffix a^{r_i} from α_i
  replace appearance of X_i in rules' bodies by $a^{\ell_j} X_j a^{r_j}$
  if $\mathrm{val}(X_i) = \epsilon$ **then** remove X_i from rules' bodies
  if there is a nonterminal transition $\delta_N(p, X_i, q)$ in N **then**
    remove transition $\delta_N(p, X_i, q)$
    if $\mathrm{val}(X_i) \neq \epsilon$ **then**
      create new states p_1, q_1 in N,
      set transitions: $\delta_N(p, a^{\ell_i}, p_1)$, $\delta_N(p_1, X_i, q_1)$, $\delta_N(q_1, a^{r_i}, q)$
    else create transition $\delta_N(p, a^{\ell_i}, q)$

Since after OutToIn a is no longer an outer letter, we may compress its maximal blocks using BlockComp. Some small twitches are needed to accommodate the a-succinct form of G and the fact that N is a-relaxed: the non-trivial part of BlockComp was the application of Lemma 2, which works for such large powers of a in npolytime, see Lemma 2. Other actions of BlockComp generalise in a simple way.

▶ **Lemma 9.** BlockComp *can be extended, so that it applies to instances satisfying (SLP 1)–(SLP 2) with G in the a-succinct form and a-relaxed-(Aut 1)–(Aut 2). The output satisfies (SLP 1)–(Aut 2) and the claim of Lemma 7 applies to such an extension.*

4.3.2 Crossing pair compression

By Lemma 9 all letters a_ℓ are inner. However, a pair of the form $a_\ell b$ can still be crossing and this can happen only when b is a left-outer letter. We try to fix it by ensuring that such a b is not a left-outer letter. To do so, we 'pop' one letter from the beginning of each nonterminal (that is, all left outer letters): let $\mathrm{val}(X_i) = bw$ for $b \in \Sigma$, we modify G so that $\mathrm{val}(X_i') = w$. Clearly, after such operations there are some (perhaps other) left-outer letters in G. Still, we show that none $a_\ell b$ is crossing.

Popping letters is performed similarly to the removal of the a-prefix, i.e. in a bottom-up fashion, starting from X_1: when considering a rule $X_i \to \alpha$, we remove the first letter from α, say b, and replace X_i by bX_i in all rules' bodies. It is easy to modify the NFA N accordingly: when there is a transition $\delta_N(p, X_i, q)$, we change it into a chain of two transitions: $\delta_{N'}(p, b, p_1)$ and $\delta_{N'}(p_1, X_i', q)$. The whole operation is not performed on X_n, as the letter \$ is not going to be compressed anyway. This description is formalised in Algorithm 5.

▶ **Lemma 10.** Pop *runs in time* polytime *and preserves (SLP 1)–(Aut 2). Let* $\mathrm{val}(X_i) = bu$, *where* $b \in \Sigma$, *then* $\mathrm{val}(X_i') = u$ *for* $i < n$ *and* $\mathrm{val}(X_n') = \mathrm{val}(X_n)$. *After running* Pop, *pairs of the form $a_\ell b$ appearing in* $\mathrm{val}(X_n)$ *are non-crossing.*

N' accepts $\mathrm{val}(X_n')$ *if and only if N accepts* $\mathrm{val}(X_n)$. *If N is deterministic, so is N'.*

Now, it is enough to apply the pair compression for non-crossing pairs to each pair of the form $a_\ell b$. For convenience, we write the whole procedure for pair compression for crossing pairs in Algorithm 6.

Algorithm 5 Pop, pops the first letter from each nonterminal

for $i \leftarrow 1 \mathinner{..} n-1$ **do** ▷ Popping letters
  let $X_i \to \alpha$ and b be the first letter of α
  remove first letter (b) from α
  replace each X_i in rules' bodies by bX_i,
  if $\alpha = \epsilon$ **then** remove X_i from rules' bodies
  if there is a transition $\delta_N(p, X_i, q)$ in N **then** ▷ NFA modification
    remove transition $\delta_N(p, X_i, q)$
    if $\alpha \neq \epsilon$ **then**
      create new state p_1 in N, set transitions: $\delta_N(p, b, p_1)$, $\delta_N(p_1, X_i, q)$
    else set transition $\delta_N(p, b, q)$

Algorithm 6 CrPairComp(a), which compresses crossing pairs $a_\ell b$

run Pop on each letter
for each a_ℓ **do**
  for each $a_\ell b$ appearing in $\mathrm{val}(X_n)$ **do**
    run PairComp($a_\ell b, c$)

▶ **Lemma 11.** CrPairComp *runs in* polytime *and preserves (SLP 1)–(Aut 2). It implements pair compression for* $a_\ell b$*, in the sense that* $\mathrm{val}(X'_i) = PC_{a_\ell b \to c}(\mathrm{val}(X_i))$ *for each* X_i*.*

N' accepts $\mathrm{val}(X'_n)$ *if and only if N accepts* $\mathrm{val}(X_n)$*. If N is deterministic, so is N'.*

4.4 Running time

Since the running time of each algorithm is npolytime, it is enough to show that the size of Σ, G and N are always polynomial in n (recall that n is unchanged throughout Algorithm 1).

▶ **Lemma 12.** *During Algorithm 1, the sizes of Σ, G, N are polynomial in n.*

Using Lemmas 6–12 it is now possible to conclude that Algorithm 1 correctly solves the FCMP for NFA, in nondeterministic polynomial (in n) time. The only source of nondeterminism is the one in Lemma 2, and so for DFA the corresponding problem can be solved deterministically.

Acknowledgements

I would like to thank Paweł Gawrychowski for introducing me to the topic and for pointing out the relevant literature.

References

1 Martin Beaudry, Pierre McKenzie, Pierre Péladeau, and Denis Thérien. Finite moniods: From word to circuit evaluation. *SIAM J. Comput.*, 26(1):138–152, 1997.

2 Paweł Gawrychowski. Optimal pattern matching in LZW compressed strings. In Dana Randall, editor, *SODA*, pages 362–372. SIAM, 2011.

3 Paweł Gawrychowski. Pattern matching in Lempel-Ziv compressed strings: fast, simple, and deterministic. In Camil Demetrescu and Magnús M. Halldórsson, editors, *ESA*, volume 6942 of *LNCS*, pages 421–432. Springer, 2011.

4 Blaise Genest and Anca Muscholl. Pattern matching and membership for hierarchical message sequence charts. *Theory of Computing Systems*, 42(4):536–567, 2008.

5 Leszek Gąsieniec, Marek Karpiński, Wojciech Plandowski, and Wojciech Rytter. Efficient algorithms for Lempel-Ziv encoding. In Rolf G. Karlsson and Andrzej Lingas, editors, *SWAT*, volume 1097 of *LNCS*, pages 392–403. Springer, 1996.

6 Artur Jeż. Faster fully compressed pattern matching by recompression. *CoRR*, 1111.3244, 2011.

7 Artur Jeż and Alexander Okhotin. One-nonterminal conjunctive grammars over a unary alphabet. *Theory of Computing Systems*, 49(2):319–342, 2011.

8 Rastislav Královič and Paweł Urzyczyn, editors. *Mathematical Foundations of Computer Science 2006, 31st International Symposium, MFCS 2006, Stará Lesná, Slovakia, August 28–September 1, 2006, Proceedings*, volume 4162 of *LNCS*. Springer, 2006.

9 Sławomir Lasota and Wojciech Rytter. Faster algorithm for bisimulation equivalence of normed context-free processes. In Královič and Urzyczyn [8], pages 646–657.

10 Yury Lifshits. Solving classical string problems an compressed texts. In Rudolf Ahlswede, Alberto Apostolico, and Vladimir I. Levenshtein, editors, *Combinatorial and Algorithmic Foundations of Pattern and Association Discovery*, volume 06201 of *Dagstuhl Seminar Proceedings*. IBFI, Schloss Dagstuhl, Germany, 2006.

11 Yury Lifshits and Markus Lohrey. Querying and embedding compressed texts. In Královič and Urzyczyn [8], pages 681–692.

12 Markus Lohrey. Word problems and membership problems on compressed words. *SIAM Journal of Computing*, 35(5):1210–1240, 2006.

13 Markus Lohrey. Compressed membership problems for regular expressions and hierarchical automata. *International Journal of Foundations of Computer Science*, 21(5):817–841, 2010.

14 Markus Lohrey and Christian Mathissen. Compressed membership in automata with compressed labels. In Alexander S. Kulikov and Nikolay K. Vereshchagin, editors, *CSR*, volume 6651 of *LNCS*, pages 275–288. Springer, 2011.

15 Markus Lohrey and Saul Schleimer. Efficient computation in groups via compression. In Volker Diekert, Mikhail V. Volkov, and Andrei Voronkov, editors, *CSR*, volume 4649 of *LNCS*, pages 249–258. Springer, 2007.

16 Jeremy MacDonald. Compressed words and automorphisms in fully residually free groups. *International Journal of Automation and Computing*, 20(3):343–355, 2010.

17 Nicolas Markey and Ph. Schnoebelen. A ptime-complete matching problem for slp-compressed words. *Inf. Process. Lett.*, 90(1):3–6, 2004.

18 Wojciech Plandowski. Testing equivalence of morphisms on context-free languages. In Jan van Leeuwen, editor, *ESA*, volume 855 of *LNCS*, pages 460–470. Springer, 1994.

19 Wojciech Plandowski. Satisfiability of word equations with constants is in PSPACE. *Journal of the ACM*, 51(3):483–496, 2004.

20 Wojciech Plandowski and Wojciech Rytter. Application of Lempel-Ziv encodings to the solution of words equations. In Kim Guldstrand Larsen, Sven Skyum, and Glynn Winskel, editors, *ICALP*, volume 1443 of *LNCS*, pages 731–742. Springer, 1998.

21 Wojciech Plandowski and Wojciech Rytter. Complexity of language recognition problems for compressed words. In Juhani Karhumäki, Hermann A. Maurer, Gheorghe Paun, and Grzegorz Rozenberg, editors, *Jewels are Forever*, pages 262–272. Springer, 1999.

Concurrency Makes Simple Theories Hard

Stefan Göller[1] and Anthony Widjaja Lin[2]

1 **Fachbereich Informatik, University of Bremen, Germany**
2 **Department of Computer Science, Oxford University, United Kingdom**

Abstract

A standard way of building concurrent systems is by composing several individual processes by product operators. We show that even the simplest notion of product operators (i.e. asynchronous products) suffices to increase the complexity of model checking simple logics like Hennessy-Milner (HM) logic and its extension with the reachability operator (EF-logic) from PSPACE to nonelementary. In particular, this nonelementary jump happens for EF-logic when we consider individual processes represented by pushdown systems (indeed, even with only one control state). Using this result, we prove nonelementary lower bounds on the size of formula decompositions provided by Feferman-Vaught (de)compositional methods for HM and EF logics, which reduce theories of asynchronous products to theories of the components. Finally, we show that the same nonelementary lower bounds also hold when we consider the relativization of such compositional methods to finite systems.

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.4.2 Grammars and Other Rewriting Systems

Keywords and phrases Modal Logic, Model Checking, Asynchronous Product, Pushdown Systems, Feferman-Vaught, Nonelementary lower bounds

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.148

1 Introduction

Concurrent systems are systems which consist of multiple processes that are simultaneously executed and possibly interacting with each other. A standard way of designing concurrent systems is to compose together several individual processes by taking some "product" operators. Various product operators have been introduced in concurrency theory and verification ranging from synchronized products (the strongest form of products) to asynchronous products (the weakest form of products). From the point of view of system design, synchronized products are the most suitable form of compositional operators. Unfortunately, from the point of view of system verification, they are known to be too powerful. For example, while reachability is NL-complete for finite transition systems, it becomes PSPACE-complete when the same problem is considered over synchronized products of finite transition systems (a.k.a. communicating finite-state machines). In the case of infinite-state systems, we see a more drastic change: while reachability is decidable in polynomial time for pushdown systems (PDS), the same problem becomes undecidable when considered over synchronized products of two PDS (note: these subsume Minsky's counter machines).

In order to circumvent the problem of high complexity and undecidability in verifying concurrent systems composed from individual processes via synchronized products, various weaker notions of products were introduced. Apart from asynchronous products which prohibit the processes to communicate, stronger product operators were introduced by restricting the types of synchronization that are allowed among the processes. Several such restrictions include bounded context switches [14], and finite synchronization [18]. These

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 148–159

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


restricted product operators can serve as good underapproximations of synchronized products. For example, a recent study of concurrency bugs conducted by the authors of [11] reveal that many real-world concurrency bugs can be detected within a small number of context switches. In addition, such restrictions also lead to decidability or lower computational complexity in model checking. For example, checking reachability over communicating finite-state machines and communicating pushdown systems with bounded context switches are both NP-complete [14].

When we consider logic model checking, the situation is not as simple. Asynchronous products do not make model checking easier than synchronized products when we use logics like LTL and CTL (and, in fact, even their restrictions to LTL(F,X) and the logic EG). Intuitively, the reason is that synchronization is easily embedded in such logics. Consequently, reachability of 2-stack pushdown systems, which is well-known to be undecidable, easily reduces to model checking any of aforementioned logics over asynchronous products of two PDS. In contrast, the situation is substantially better when we consider simpler logics like Hennessy-Milner (HM) Logic and its extension with the reachability operator (i.e. EF-logic). In fact, powerful *(Feferman-Vaught) compositional methods* (e.g. [12, 15, 18]), which reduce model checking of product structures to model checking of their components, can be used for obtaining decidability or better upper complexity bounds of model checking HM-logic and EF-logic. We now state the most basic form of such compositional methods from [15].

▶ **Theorem 1** ([15]). *For each HM/EF formula φ over the action labels $\mathbb{A} = \mathbb{A}_1 \cup \ldots \cup \mathbb{A}_k$, for nonempty disjoint sets $\mathbb{A}_1, \ldots, \mathbb{A}_k$, one can compute k finite sets of HM/EF formulas $\{\psi_i^1\}_{i \in I_1}, \ldots, \{\psi_i^k\}_{i \in I_k}$ over $\mathbb{A}_1, \ldots, \mathbb{A}_k$ respectively, and a positive boolean formula (i.e. no negations) β with variables $\{x_i^1\}_{i \in I_1}, \ldots, \{x_i^k\}_{i \in I_k}$ such that for* all *transition systems $\mathcal{T}_1, \ldots, \mathcal{T}_k$ with initial states $s_1, \ldots, s_k$ we have $(\prod_{i=1}^k \mathcal{T}_i, \bar{s}) \models \varphi$ if and only if $\beta[\mu]$ is true, where $\bar{s} = (s_1, \ldots, s_k)$ and μ assigns the variables of β as follows: $\mu(x_i^j) = 1$ if and only if $(\mathcal{T}_j, s_j) \models \psi_i^j$.*

Actually, a stronger version of Theorem 1 was proven in [15] (e.g. with atomic propositions). In the statement of Theorem 1, the k sets of formulas and the positive boolean formula β are referred to as the *decomposition* of φ. To give some concrete illustrations of the power of this compositional theorem, Theorem 1 can be used to show that model-checking *fixed* EF formulas (i.e. the complexity is only measured the size of the system) is: NL-complete for asynchronous products of finite systems (c.f. PSPACE-completeness of communicating finite-state systems), PSPACE-complete for asynchronous products of PDS [16], and P-complete for asynchronous products of Basic Process Algebras (equivalently, one-state PDS).

Despite the aforementioned usefulness of Feferman-Vaught compositional methods, the technique yields algorithms with nonelementary complexity in the size of the formula (see [15]), which is not desirable from both theoretical and practical viewpoints. In fact, it was recently shown that when we consider stronger logics like first-order logic, where Feferman-Vaught compositional methods are also available (e.g. see [12]), this nonelementary complexity is unavoidable [6]. It is natural to ask whether this nonelementary complexity is avoidable when we consider simpler logics like HM-logic or EF-logic. In fact, this open question has been posed in the literature (e.g. [7, 15]).

This open question actually brings us to a more fundamental open question: how do asynchronous products affect the complexity of model checking of HM-logic and EF-logic? This open question has manifested itself in the literature in various concrete forms. As an example, take the result that model checking EF-logic over pushdown systems is PSPACE-complete [20]. Over asynchronous products of *two* pushdown systems, the best algorithm for model checking EF-logic runs in nonelementary time [16]. In fact, the same nonelementary

gap is currently present for asynchronous products of two BPAs. Failing to answer this open question is also the reason for the existing nonelementary complexity gaps for several verification problems for *PA-processes* [13], which are an extension of asynchronous products of BPAs with process creations.

Contributions. In this paper, we provide answers to the above open questions. A main contribution of this paper is to show that, for each integer $k > 0$, there exists an asynchronous product of two BPAs whose EF-logic theory requires k-fold exponential time to solve. This means that model checking EF-logic over the class of asynchronous products of two BPAs requires nonelementary time, which is in stark contrast to PSPACE-completeness of EF model checking over BPAs [20]. As an upshot of our result, it follows that model checking EF-logic over PA-processes requires nonelementary time, which solves the open question posed by R. Mayr [13].

We also show that similar results hold for HM-logic. More precisely, we prove that for each integer $k > 0$ there exists an asynchronous product of two *prefix-recognizable systems* (an extension of BPAs introduced by Caucal [5] by allowing infinitely many rewrite rules compactly represented by regular languages) whose HM-logic theory requires k-fold exponential time to solve. This means that model checking HM-logic over the class of asynchronous products of two prefix-recognizable systems requires nonelementary time, which is in stark contrast to PSPACE-completeness of HM-logic model checking over prefix-recognizable systems (which easily follows[1] from the result of [20]).

An important corollary of our two aforementioned results is that there is no elementary algorithm for computing decompositions of formulas in HM-logic and EF-logic in the sense of Theorem 1. We go one step further to show that no decompositions of formulas in HM-logic and EF-logic of elementary size even exist in general. In other words, both descriptional and computational complexity of compositional methods of HM-logic and EF-logic in the sense of Theorem 1 are inherently nonelementary. Incidentally, this also entails the same nonelementary lower bounds for compositional methods provided in [7] since they generalize Theorem 1.

So far, our nonelementary lower bounds for compositional methods for HM-logic and EF-logic require the use of infinite-state systems. This still leaves the possibility that Theorem 1 could hold when we restrict the transition systems under consideration to be finite-state. Questions of this form are of particular interests in finite model theory (e.g. see [10]) and in verification of finite-state systems. We show, however, that the same nonelementary lower bounds relativize to the class of asynchronous products of finite systems. Whether the nonelementary lower bounds hold when relativized to the class of asynchronous products of *finite trees* is left for future work.

2 Preliminaries

General: By $\mathbb{N} = \{0, 1, \ldots\}$ we denote the set of *nonnegative integers*. For each $i, j \in \mathbb{N}$, we define the interval $[i, j] = \{i, i+1, \ldots, j\}$. Let $f : \mathbb{N} \to \mathbb{N}$ be a function. We write $f(n) = \mathsf{poly}(n)$ if there is some polynomial $p(n)$ such that $f(n) \leq p(n)$ for all $n \in \mathbb{N}$. We write $f(n) = \exp(n)$ if there is some polynomial $p(n)$ such that $f(n) \leq 2^{p(n)}$ for all $n \in \mathbb{N}$. We define the standard *Tower function* $\mathsf{Tower} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ inductively as $\mathsf{Tower}(0, n) = n$ and $\mathsf{Tower}(k, n) = 2^{\mathsf{Tower}(k-1,n)}$, for each $k > 0$ and each $n \in \mathbb{N}$.

[1] On the same note, even μ-calculus over prefix-recognizable systems is only EXP-complete [9]

Automata: A *deterministic finite automaton (DFA)* is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, where Q is a finite set of *states*, Σ is a finite *alphabet*, $q_0 \in Q$ is the *initial state*, $\delta : Q \times \Sigma \to Q$ is the *transition function*, and $F \subseteq Q$ is the set of *final states*. By $L(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ accepts } w\}$ we denote the *language* of $\mathcal{A}$. The *size* of $\mathcal{A}$ is defined as $|\mathcal{A}| = |Q|$.

Systems: Let us fix a countable set of action labels Act. A *transition system* is tuple $\mathcal{T} = (S, \mathbb{A}, \{\xrightarrow{a} \mid a \in \mathbb{A}\})$, where S is a set of *states*, $\mathbb{A} \subseteq \mathsf{Act}$ is a finite set of action labels, and where $\xrightarrow{a} \subseteq S \times S$ is a set of *transitions* for each $a \in \mathbb{A}$. We say $\mathcal{T}$ is *finite* if S is finite. A *pointed transition system* is a pair $(\mathcal{T}, s)$, where $\mathcal{T}$ is a transition system and s is state of $\mathcal{T}$. We write $s \xrightarrow{a} t$ to abbreviate $(s,t) \in \xrightarrow{a}$. We apply similar abbreviations for other binary relations over S. For each $\Gamma \subseteq \mathbb{A}$, we define $\xrightarrow{\Gamma} = \bigcup_{a \in \Gamma} \xrightarrow{a}$.

Given $k \geq 1$ transition systems $\mathcal{T}_1 = (S_1, \mathbb{A}_1, \{\xrightarrow{a}_1 \mid a \in \mathbb{A}_1\}), \ldots, \mathcal{T}_k = (S_k, \mathbb{A}_k, \{\xrightarrow{a}_k \mid a \in \mathbb{A}_k\})$, where the $\mathbb{A}_i \cap \mathbb{A}_j = \emptyset$ for each $i \neq j$, we define its *asynchronous product* $\prod_{i=1}^k \mathcal{T}_i = (S, \mathbb{A}, \{\xrightarrow{a} \mid a \in \mathbb{A}\})$, where $S = \prod_{i=1}^k S_i$, $\mathbb{A} = \bigcup_{i=1}^k \mathbb{A}_i$, and where for each $a \in \mathbb{A}$ we have $(s_1, \ldots, s_k) \xrightarrow{a} (s'_1, \ldots, s'_k)$ if and only if $s_i \xrightarrow{a}_i s'_i$ for some $i \in [1,k]$ with $a \in \mathbb{A}_i$ and $s_j = s'_j$ for each $j \in [1,k] \setminus \{i\}$.

Logic: Formulas of EF-logic over a finite set $\mathbb{A} \subseteq \mathsf{Act}$ of labels are given by the following grammar $\varphi ::= \top \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\Gamma\rangle\varphi \mid \langle\Gamma^*\rangle\varphi$, where $\Gamma \subseteq \mathbb{A}$:
We introduce the usual abbreviations $\bot = \neg\top$, $\varphi_1 \vee \varphi_2 = \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $\varphi_1 \to \varphi_2 = \neg\varphi_1 \vee \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2 = (\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)$, $[\Gamma]\varphi = \neg\langle\Gamma\rangle\neg\varphi$, and $[\Gamma^*]\varphi = \neg\langle\Gamma^*\rangle\neg\varphi$. We also write $\langle\Gamma\rangle^n$ (resp. $[\Gamma]^n$) as an abbreviation for a sequence of n consecutive $\langle\Gamma\rangle$'s (resp. $[\Gamma]$'s). By $|\varphi|$ we denote the *size* of each EF formula $|\varphi|$ defined as usually. *Hennessy-Milner logic (*HM*)* is the syntactic fragment of EF that one obtains by forbidding formulas of the kind $\langle\Gamma^*\rangle\varphi$. Since we allow formulas of the form $\langle\Gamma^*\rangle\varphi$ for subsets Γ of the action labels (instead of only $\langle\mathbb{A}^*\rangle$), our version of EF is slightly more general than the standard definition of EF-logic. However, our results easily carry over to the standard definition of EF-logic.

For each transition system $\mathcal{T} = (S, \mathbb{A}, \{\xrightarrow{a} \mid a \in \mathbb{A}\})$ and each EF-formula φ (over $\mathbb{A}$) define the set of states $[\![\varphi]\!]_\mathcal{T} \subseteq S$ that satisfy φ by induction on the structure of φ as follows:

$$
\begin{array}{lcllcl}
[\![\top]\!]_\mathcal{T} & = & S & [\![\langle\Gamma\rangle\varphi]\!]_\mathcal{T} & = & \{s \in S \mid \exists t \in [\![\varphi]\!]_\mathcal{T} : s \xrightarrow{\Gamma} t\} \\
[\![\neg\varphi]\!]_\mathcal{T} & = & S \setminus [\![\varphi]\!]_\mathcal{T} & [\![\langle\Gamma^*\rangle\varphi]\!]_\mathcal{T} & = & \{s \in S \mid \exists t \in [\![\varphi]\!]_\mathcal{T} : s \xrightarrow{\Gamma}{}^* t\} \\
[\![\varphi_1 \wedge \varphi_2]\!]_\mathcal{T} & = & [\![\varphi_1]\!]_\mathcal{T} \cap [\![\varphi_2]\!]_\mathcal{T} & & &
\end{array}
$$

We also write $(\mathcal{T}, s) \models \varphi$ whenever $s \in [\![\varphi]\!]_\mathcal{T}$ or simply $s \models \varphi$ when $\mathcal{T}$ is clear from the context.

Infinite-state models: A *pushdown system (PDS)* is a tuple $\mathcal{P} = (\Sigma, \mathbb{A}, \Delta)$, where Σ is a finite set of *process constants*, $\mathbb{A} \subseteq \mathsf{Act}$ is a finite set of action labels and Δ is a finite set of *rewrite rules* of the form $u \mapsto_a v$, where $a \in \mathbb{A}$, $u \in \Sigma^*$ and $v \in \Sigma^*$. A *basic process algebra (BPA)* is PDA $\mathcal{P} = (\Sigma, \mathbb{A}, \Delta)$, where for each $u \mapsto_a v \in \Delta$ we have $|u| = 1$. The associated transition system $\mathcal{T}(\mathcal{P})$ is defined as $\mathcal{T}(\mathcal{P}) = (\Sigma^*, \mathbb{A}, \{\xrightarrow{a} \mid a \in \mathbb{A}\})$, where $\xrightarrow{a} = \{(uw, vw) \mid u \mapsto_a v \in \Delta, w \in \Sigma^*\}$ for each $a \in \mathbb{A}$. The *size* of a PDS is defined as $|\mathcal{P}| = |\Sigma| + |\mathbb{A}| + \sum_{u \mapsto_a v \in \Delta}(|u| + |v|)$.

3 Hardness of asynchronous product

We start by proving a nonelementary lower bound for the problem of *model checking* EF *on BPA×BPA*: given two BPAs $\mathcal{P} = (\Sigma, \mathbb{A}, \Delta)$, $\mathcal{P}' = (\Sigma', \mathbb{A}', \Delta')$ with $\mathbb{A} \cap \mathbb{A}' = \emptyset$, a pair of process constants $\langle X, X' \rangle \in \Sigma \times \Sigma'$, and an EF formula φ over $\mathbb{A} \cup \mathbb{A}'$, check whether $(\mathcal{T}(\mathcal{P}) \times \mathcal{T}(\mathcal{P}'), \langle X, X' \rangle) \models \varphi$.

▶ **Theorem 2.** *Model checking EF on BPA×BPA is nonelementary.*

We then show that this lower bound implies a nonelementary lower bound for model checking HM-logic over the class of asynchronous products of two prefix-recognizable systems (a precise definition is given below).

3.1 Proof of Theorem 2

The structure of the proof of Theorem 2 is as follows. We first show how to encode large counters as EF formulas evaluated over the class of asynchronous products of two BPAs. Such large counters are enforced by the two stacks in the two BPAs, which alternately "guess" an encoding of a counter and "check" the correctness of the encoding. As we will see later, this encoding of large counters can be used to encode memberships of $\mathsf{Tower}(k, cn)$ space-bounded Turing machines for any fixed $k > 0$.

Large counters: The following encoding of large numbers is from [19, 3]. In the following, the notations n and ℓ will range over $\mathbb{N}$. We define the alphabets $\Omega_\ell = \{0_\ell, 1_\ell\}$ and the values $\mathsf{val}(0_\ell) = 0$ and $\mathsf{val}(1_\ell) = 1$ for each $\ell \geq 0$.

A $(1, n)$-*counter* is a word from Ω_0^n. The *value* $\mathsf{val}(c)$ of some $(1, n)$-counter $c = \sigma_0 \cdots \sigma_{n-1}$ is defined as $\mathsf{val}(c) = \sum_{i=0}^{n-1} 2^i \cdot \mathsf{val}(\sigma_i) \in [0, 2^n - 1]$. So the set of values $\mathsf{val}(c)$ for $(1, n)$-counters c equals $[0, 2^n - 1] = [0, \mathsf{Tower}(1, n) - 1]$. An (ℓ, n)-*counter* with $\ell > 1$ is a word $c = c_0\sigma_0c_1\sigma_1 \dots c_m\sigma_m$, where $m = \mathsf{Tower}(\ell - 1, n) - 1$, each c_i is an $(\ell - 1, n)$-counter with $\mathsf{val}(c_i) = i$ and $\sigma_i \in \Omega_{\ell-1}$ for each $i \in [0, m]$. We define $\mathsf{val}(c) = \sum_{i=0}^{m} 2^i \cdot \mathsf{val}(\sigma_i)$. Observe that $\mathsf{val}(c) \in [0, \mathsf{Tower}(\ell, n) - 1]$ and the length of each (ℓ, n)-counter is uniquely determined by ℓ and n.

In the following, we define $\Omega'_\ell = \{0'_\ell, 1'_\ell\}$ to be a fresh copy of Ω_ℓ; moreover define $\Sigma_\ell = \bigcup_{i=0}^{\ell} \Omega_i$ and analogously $\Sigma'_\ell = \bigcup_{i=0}^{\ell} \Omega'_i$.

Definition of the two BPAs: For each integer $\ell > 0$, let us define the following simple BPAs $\mathcal{P}_\ell = (\Sigma_\ell, \mathbb{L}_\ell, \Delta_\ell)$, where

- $\mathbb{L}_\ell = \Sigma_\ell \cup \overline{\Sigma_\ell}$, where $\overline{\Sigma_\ell} = \{\overline{\sigma} \mid \sigma \in \Sigma_\ell\}$ is a dual copy of Σ_ℓ.
- $\Delta_\ell = \{\tau \mapsto_\sigma \sigma\tau \mid \sigma, \tau \in \Sigma_\ell\} \cup \{\sigma \mapsto_{\overline{\sigma}} \varepsilon \mid \sigma \in \Sigma_\ell\}$.

The transition system $\mathcal{T}(\mathcal{P}_\ell)$ has a fairly regular behavior. The set of states is Σ_ℓ^*. Executing an action $\overline{\sigma} \in \overline{\Sigma_\ell}$ from a state $u \in (\Sigma_\ell)^*$ allows to remove exactly this leftmost symbol σ from u if u begins with σ, otherwise $\overline{\sigma}$ cannot be executed from u. Dually, from every nonempty state $u \in (\Sigma_\ell)^+$ of $\mathcal{T}(\mathcal{P}_\ell)$ we can execute every action $\sigma \in \Sigma_\ell$ yielding the state σu; the only state from which the $\sigma \in \Sigma_\ell$ are not executable is the empty word ε. We define the BPA $\mathcal{P}'_\ell$ analogously to $\mathcal{P}_\ell$ but by priming every symbol. Formally, $\mathcal{P}'_\ell = (\Sigma'_\ell, \mathbb{L}'_\ell, \Delta'_\ell)$, where

- $\mathbb{L}'_\ell = \Sigma'_\ell \cup \overline{\Sigma'_\ell}$, where $\overline{\Sigma'_\ell} = \{\overline{\sigma'} \mid \sigma' \in \Sigma'_\ell\}$ is a dual copy of Σ'_ℓ.
- $\Delta'_\ell = \{\tau' \mapsto_{\sigma'} \sigma'\tau' \mid \sigma', \tau' \in \Sigma'_\ell\} \cup \{\sigma' \mapsto_{\overline{\sigma'}} \varepsilon \mid \sigma' \in \Sigma'_\ell\}$.

Note that the set of states of $\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell)$ is $(\Sigma_\ell)^* \times (\Sigma'_\ell)^*$. Given a state $s = (u, u') \in (\Sigma_\ell)^* \times (\Sigma'_\ell)^*$, we call u the *left stack of* s and u' the *right stack of* s. So we treat the words u and u' as stacks with their left-most symbols being the top of the stack. Recall that every (ℓ, n)-counter is in particular a word over $\Sigma_{\ell-1}$. We extend this notion to words over $\Sigma'_{\ell-1}$ in the usual way. So each (ℓ, n)-counter will in particular be either a word over $\Sigma_{\ell-1}$ or over $\Sigma'_{\ell-1}$, depending on whether we address the left stack or the right stack. Note that if some word over Σ_k (resp. over Σ'_k) has an (ℓ, n)-counter as a prefix, then the length of this prefix is uniquely determined by ℓ and n, namely $\mathsf{Tower}(\ell - 1, n)$. An *extended* (ℓ, n)-*counter* is either a string $c\sigma$, where either $c \in \Sigma_{\ell-1}^*$ is an (ℓ, n)-counter and $\sigma \in \Omega_\ell$, or a string $c'\sigma'$, where $c' \in (\Sigma'_{\ell-1})^*$ is an (ℓ, n)-counter and $\sigma' \in \Omega'_\ell$.

Next, we define some EF formulas (with primed counterparts for the right stack) for each $\ell, n \in \mathbb{N}$:

1. $\mathsf{count}^{\sigma}_{(\ell,n)}$ for each $\sigma \in \Omega_\ell$ such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{count}^{\sigma}_{(\ell,n)}$ if and only if for some (ℓ, n)-counter c we have $c\sigma$ is a prefix of u.
2. $\mathsf{count}^{\sigma'}_{(\ell,n)}$ for each $\sigma' \in \Omega_\ell$ such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{count}^{\sigma'}_{(\ell,n)}$ if and only if for some (ℓ, n)-counter c' we have $c'\sigma'$ is a prefix of u'.
3. $\mathsf{xcount}_{(\ell,n)}$ such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{xcount}_{(\ell,n)}$ if and only if some extended (ℓ, n)-counter $c\sigma$ (for $\sigma \in \Omega_\ell$) is a prefix of u.
4. $\mathsf{xcount}'_{(\ell,n)}$ such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{xcount}'_{(\ell,n)}$ if and only if some extended (ℓ, n)-counter $c'\sigma'$ (for $\sigma' \in \Omega'_\ell$) is a prefix of u'.
5. $\mathsf{first}_{(\ell,n)}$ (resp. $\mathsf{first}'_{(\ell,n)}$) such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{first}_{(\ell,n)}$ (resp. $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{first}'_{(\ell,n)}$) if and only if some extended (ℓ, n)-counter $c\sigma$ (resp. $c'\sigma'$) with $\mathsf{val}(c) = 0$ (resp. $\mathsf{val}(c') = 0$) is a prefix of u (resp. u').
6. $\mathsf{last}_{(\ell,n)}$ (resp. $\mathsf{last}'_{(\ell,n)}$) such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{last}_{(\ell,n)}$ (resp. $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{last}'_{(\ell,n)}$) if and only if some extended (ℓ, n)-counter $c\sigma$ (resp. $c'\sigma'$) with $\mathsf{val}(c) = \mathsf{Tower}(\ell, n) - 1$ (resp. $\mathsf{val}(c') = \mathsf{Tower}(\ell, n) - 1$) is a prefix of u (resp. u').
7. $\mathsf{eq}_{(\ell,n)}$ such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{eq}_{(\ell,n)}$ if and only if there exist extended (ℓ, n)-counters $c\sigma \in (\Sigma_{\ell-1})^*\Omega_\ell$ and $c'\sigma' \in (\Sigma'_{\ell-1})^*\Omega'_\ell$ such that (i) $c\sigma$ is a prefix of u, (ii) $c'\sigma'$ is a prefix of u', and (iii) $\mathsf{val}(c) = \mathsf{val}(c')$.
8. $\mathsf{inc}_{(\ell,n)}$ (resp. $\mathsf{inc}'_{(\ell,n)}$) such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{inc}_{(\ell,n)}$ (resp. $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{inc}'_{(\ell,n)}$) if and only if there exist extended (ℓ, n)-counters $c\sigma \in (\Sigma_{\ell-1})^*\Omega_\ell$ and $c'\sigma' \in (\Sigma'_{\ell-1})^*\Omega'_\ell$ such that (i) $c\sigma$ is a prefix of u, (ii) $c'\sigma'$ is a prefix of u', and (iii) $\mathsf{val}(c) + 1 = \mathsf{val}(c')$ (resp. $\mathsf{val}(c') + 1 = \mathsf{val}(c)$).
9. $\mathsf{succ}_{(\ell,n)}$ (resp. $\mathsf{succ}'_{(\ell,n)}$) such that $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{succ}_{(\ell,n)}$ (resp. $(\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{succ}'_{(\ell,n)}$) if and only if there are extended (ℓ, n)-counters $c_1\sigma_1$ and $c_2\sigma_2$ (resp. $c'_1\sigma'_1$ and $c'_2\sigma'_2$) with $\sigma_1, \sigma_2 \in \Omega_\ell$ (resp. $\sigma'_1, \sigma'_2 \in \Omega'_\ell$) such that $c_1\sigma_1c_2\sigma_2$ is a prefix of u and $\mathsf{val}(c_1) + 1 = \mathsf{val}(c_2)$ (resp. $\mathsf{val}(c'_1) + 1 = \mathsf{val}(c'_2)$).

The formulas that we will define will be exponential in ℓ and polynomial in n (represented in unary). This definition will be given by induction on ℓ. We will start with the following simple observations: $\mathsf{xcount}_{(\ell,n)} = \bigvee_{\sigma\in\Omega_\ell} \mathsf{count}^{\sigma}_{(\ell,n)}$, and $\mathsf{xcount}'_{(\ell,n)} = \bigvee_{\sigma'\in\Omega'_\ell} \mathsf{count}^{\sigma'}_{(\ell,n)}$. We will now construct several formulas φ that we evaluate on $\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell)$ expressing properties of the *left stack*. Without making them explicit, we can construct corresponding analogs φ' expressing the respective property on the *right stack*.

Let us proceed by defining the above formulas for the case of $\ell = 1$. We define $\mathsf{count}^{\sigma}_{(1,n)}$ and $\mathsf{count}^{\sigma'}_{(1,n)}$ as $\mathsf{count}^{\sigma}_{(1,n)} = \langle\overline{\Omega_0}\rangle^n \langle\overline{\sigma}\rangle\top$ and $\mathsf{count}^{\sigma'}_{(1,n)} = \left\langle\overline{\Omega'_0}\right\rangle^n \langle\overline{\sigma'}\rangle\top$. We put $\mathsf{first}_{(1,n)} = \langle\overline{0_0}\rangle^n\langle\overline{\Omega_1}\rangle\top$. The definition of $\mathsf{last}_{(1,n)}$ is analogous. We also define $\mathsf{eq}_{(1,n)}$ as $\mathsf{xcount}_{(1,n)} \wedge \mathsf{xcount}'_{(1,n)} \wedge \bigwedge_{i=0}^{n-1} \langle\overline{\Omega_0}\rangle^i\langle\overline{\Omega'_0}\rangle^i \wedge \bigwedge_{\sigma\in\Omega_0}(\langle\overline{\sigma}\rangle\top \leftrightarrow \langle\overline{\sigma'}\rangle\top)$. The definitions of $\mathsf{inc}_{(1,n)}$ and $\mathsf{succ}_{(1,n)}$ are analogous.

Let us now proceed to the case of $\ell > 1$. We start by defining the formula $\mathsf{count}^{\sigma}_{(\ell,n)}$ for each $\sigma \in \Omega_\ell$. We will achieve this, by making use of the formulas $\mathsf{first}_{(\ell-1,n)}$, $\mathsf{last}_{(j,n)}$ with $j \in [1, \ell-1]$, $\mathsf{xcount}_{(\ell-1,n)}$, and $\mathsf{succ}_{(\ell-1,n)}$. The first two conjuncts of the definition of $\mathsf{count}^{\sigma}_{(\ell,n)}$ are self-explanatory,

$$\mathsf{count}^{\sigma}_{(\ell,n)} \;=\; \mathsf{first}_{(\ell-1,n)} \wedge \left[\overline{\Sigma_{\ell-1}}^*\right] \left(\mathsf{xcount}_{(\ell-1,n)} \rightarrow \left(\mathsf{last}_{(\ell-1,n)} \vee \mathsf{succ}_{(\ell-1,n)}\right)\right) \wedge \mathsf{add}^{\sigma},$$

whereas the formula add^σ will express that the symbol σ follows after the top-most (ℓ, n)-counter. Formally we put $\mathsf{add}^\sigma = \psi^\sigma_{\ell-1}$, where

$$\psi^\sigma_j = \begin{cases} \left[\overline{\Sigma_j}^*\right] \left(\mathsf{last}_{(j,n)} \to \psi^\sigma_{j-1}\right) & \text{if } j > 1 \\ \left[\overline{\Sigma_1}^*\right] \left(\mathsf{last}_{(1,n)} \to \langle \overline{1_0} \rangle^n \langle \overline{1_1} \rangle \langle \overline{1_2} \rangle \cdots \langle \overline{1_{\ell-2}} \rangle \langle \overline{\Omega_{\ell-1}} \rangle \langle \overline{\sigma} \rangle \top\right) & \text{if } j = 1. \end{cases}$$

Intuitively, the formula $\psi^\sigma_{\ell-1}$ jumps to last extended $(1, n)$-counter of the last extended $(2, n)$-counter ... of the last extended $(\ell - 1, n)$-counter and expresses that the correct sequence follows from this position.

Define $\mathsf{first}_{(\ell,n)}$ as $\mathsf{first}_{(\ell,n)} = \mathsf{xcount}_{(\ell,n)} \wedge \left[\overline{\Sigma_{\ell-1}}^*\right] \left(\langle \overline{\Omega_{\ell-1}} \rangle \top \to \langle \overline{0_{\ell-1}} \rangle \top\right)$, similarly we define $\mathsf{last}_{(\ell,n)}$. We set $\mathsf{eq}_{(\ell,n)}$ as the conjunction of $\mathsf{xcount}_{(\ell,n)} \wedge \mathsf{xcount}'_{(\ell,n)}$ and

$$\left[\overline{\Sigma_{\ell-1}}^*\right] \left(\mathsf{xcount}_{(\ell-1,n)} \to \left(\left\langle \overline{\Sigma'_{\ell-1}}^* \right\rangle \left(\mathsf{eq}_{(\ell-1,n)} \wedge \bigwedge_{\sigma \in \Omega_{\ell-1}} (\langle \overline{\Sigma^*_{\ell-2}} \rangle \langle \overline{\sigma} \rangle \top \leftrightarrow \langle \overline{\Sigma'^*_{\ell-2}} \rangle \langle \overline{\sigma'} \rangle \top)\right)\right)\right).$$

Let us give some intuition on the formulas $\mathsf{eq}_{(\ell,n)}$ for each $\ell \in [2, k]$: Whenever we pop from the left stack some string from $(\Sigma_{\ell-1})^*$ until on top of the left stack there is some extended $(\ell - 1, n)$-counter $c\sigma$, one can remove from the right stack a string from $(\Sigma'_{\ell-1})^*$ yielding an extended $(\ell - 1, n)$ counter $c'\tau'$ on top of the right stack such that $\mathsf{val}(c) = \mathsf{val}(c')$ and moreover $\sigma = \tau$ holds.

In analogy to $\mathsf{eq}_{(\ell,n)}$ one can define the formula $\mathsf{inc}_{(\ell,n)}$. Finally, let us define $\mathsf{succ}_{(\ell,n)}$. We put

$$\mathsf{succ}_{(\ell,n)} \;=\; \langle \Sigma'_\ell \rangle \left\langle (\Sigma'_{\ell-1})^* \right\rangle \left(\mathsf{eq}_{(\ell,n)} \wedge \left\langle \overline{\Sigma_{\ell-1}}^* \right\rangle \langle \overline{\Sigma_\ell} \rangle \, \mathsf{inc}'_{(\ell,n)}\right)$$

Intuitively, we the formula $\mathsf{succ}_{(\ell,n)}$ pushes onto the right stack some string that it checks to be a copy of the topmost extended (ℓ, n)-counter of the left stack via $\mathsf{eq}_{(\ell,n)}$, then pops the topmost extended (ℓ, n)-counter of the left stack and then invokes the formula $\mathsf{inc}'_{(\ell,n)}$.

It is easy to see that the formulas given above express the desired properties. Furthermore, we note that the size of each formula is exponential in ℓ and polynomial in n.

By using standard arguments (e.g. see the proof of PSPACE-hardness of EF model checking over pushdown systems in [2]), we can now complete the proof of Theorem 2. In the following, we shall only provide a sketch. For each integer $k > 0$, there exists a fixed Turing machine $\mathcal{M}$ operating with $\mathsf{Tower}(k - 1, cn)$ space (for a constant c) whose membership problem is complete for $\mathsf{SPACE}(\mathsf{Tower}(k - 1, \mathsf{poly}(n)))$. Each such membership problem can easily be reduced to EF model checking over the class of asynchronous products of two BPAs in polynomial time as follows. For an input word w of $\mathcal{M}$ of length n one can construct formulas $\mathsf{xcount}_{(k,cn)}$ and the pair of BPAs $\mathcal{P}_{k+1}$ and $\mathcal{P}'_{k+1}$ in time $\mathsf{poly}(n)$. Each computation of $\mathcal{M}$ can be viewed as a sequence of configurations (each being a (k, cn)-counters), which when considered together is also a $(k + 1, cn)$-counter, satisfying the transition conditions of $\mathcal{M}$ (e.g. two consecutive configurations respect the transition function of $\mathcal{M}$). One can express the computation of $\mathcal{M}$ on w by an EF formula of the kind $\langle \Sigma^*_{k+1} \rangle \varphi$, where $\langle \Sigma^*_{k+1} \rangle$ aims at pushing a sequence onto the left stack and where φ expresses that this sequence expresses the desired properties.

We will make use of the following lemma in Section 4.

▶ **Lemma 3.** *Every DFA accepting the regular language*

$$L_{\ell,n} \;=\; \{u \in \Sigma^*_\ell \mid \exists u' \in (\Sigma'_\ell)^* : (\mathcal{T}(\mathcal{P}_\ell) \times \mathcal{T}(\mathcal{P}'_\ell), (u, u')) \models \mathsf{xcount}_{(\ell,n)}\}$$

has at least $\mathsf{Tower}(\ell - 1, n) + 1$ *states.*

Proof. Recall that every extended (ℓ, n)-counter has length exactly $\mathsf{Tower}(\ell - 1, n) + 1$. We have $L_{\ell,n} = \{c\sigma w \mid w \in \Sigma_\ell^*, c\sigma \text{ is some extended } (\ell, n)\text{-counter}\}$. The corollary now follows from the following simple observation: Every DFA $\mathcal{A}$ over some alphabet Σ with $L(\mathcal{A}) = U \cdot \Sigma^*$ for some $\emptyset \subsetneq U \subseteq \Sigma^m$ has at least m states. ◀

3.2 Lower bounds for HM-logic

We conclude this section by showing how Theorem 2 implies a nonelementary lower bound for model checking HM on the asynchronous product of two prefix-recognizable systems. A *prefix-recognizable system* is a tuple $\mathcal{R} = (\Sigma, \mathbb{A}, \Delta)$, where Σ is finite set of process constants, $\mathbb{A} \subseteq \mathsf{Act}$ is a finite set of action labels and Δ is a finite set of rewrite rules of the form $U \mapsto_a V$, where $a \in \mathbb{A}$, and where $U, V \subseteq \Sigma^*$ are regular languages given as DFAs, say. The associated transition system is $\mathcal{T}(\mathcal{R}) = (\Sigma^*, \mathbb{A}, \{\xrightarrow{a} \mid a \in \mathbb{A}\})$, where $\xrightarrow{a} = \{(uw, vw) \mid u \in U, v \in V, w \in \Sigma^* \text{ for some rule } U \mapsto_a V \in \Delta\}$ for each $a \in \mathbb{A}$.

Remark: One can construct from a given pair of BPAs $\mathcal{P} = (\Sigma, \mathbb{A}, \Delta)$ and $\mathcal{P}' = (\Sigma', \mathbb{A}', \Delta')$ and a given EF formula φ over $\mathbb{A} \cup \mathbb{A}'$ a pair of prefix-recognizable systems $\mathcal{R} = (\Sigma, \mathbb{A}, \Delta_{\mathcal{R}})$ and $\mathcal{R}' = (\Sigma', \mathbb{A}', \Delta'_{\mathcal{R}})$ and some HM formula $\widetilde{\varphi}$ such that $[\![\varphi]\!]_{\mathcal{T}(\mathcal{P})\times\mathcal{T}(\mathcal{P})} = [\![\widetilde{\varphi}]\!]_{\mathcal{T}(\mathcal{R})\times\mathcal{T}(\mathcal{R}')}$ as follows: By [4] one can compute for each $\Gamma \subseteq \mathbb{A}$ (analogously for each $\Gamma' \subseteq \mathbb{A}'$) a pair regular languages U_Γ and V_Γ (resp. $U_{\Gamma'}$ and $V_{\Gamma'}$) each accepted by DFAs of at most exponential size such that the relation $\xrightarrow{\Gamma}^*$ over $\Sigma^* \times \Sigma^*$ (resp. $\xrightarrow{\Gamma'}^*$ over $(\Sigma')^* \times (\Sigma')^*$) is exactly $\mathcal{R}(\widetilde{\Gamma}) = \{(uw, vw) \mid u \in U_\Gamma, v \in V_\Gamma, w \in \Sigma^*\}$ (resp. $\mathcal{R}'(\widetilde{\Gamma'}) = \{(uw, vw) \mid u \in U_{\Gamma'}, v \in V_{\Gamma'}, w \in (\Sigma')^*\}$). The latter is even shown for PDAs in [4]. Hence we can define the HM formula $\widetilde{\varphi}$ to emerge from φ by replacing each occurence of $\langle\Gamma^*\rangle$ by $\langle\widetilde{\Gamma}\rangle$ and each occurrence of $\langle(\Gamma')^*\rangle$ by $\langle\widetilde{\Gamma'}\rangle$.

Theorem 2 and the previous remark immediately imply the following corollary.

▶ **Corollary 4.** *Model checking* HM *on the asynchronous product of two prefix-recognizable systems is nonelementary.*

We remark that model checking HM on a single prefix-recognizable system is only PSPACE-complete; the upper bound can be shown via reduction to EF model checking pushdown systems, which is in PSPACE by [20].

4 Lower bounds for compositional methods for HM and EF logics

We start by proving nonelementary lower bounds for Feferman-Vaught type of compositional methods for HM and EF logics (i.e. Theorem 1) already over the the class of asynchronous products of *two* transition systems. In Section 4.2 we will then show how our lower bounds can be relativized to the class of all asynchronous products of *two finite* transition systems.

Let us briefly recall decompositions following Theorem 1 for EF logic of asynchronous products of two transition systems. Analogously HM can be dealt with. A *decomposition* of the asynchronous product of two transition systems, the first component being defined over action labels $\mathbb{A}$ and the second one over $\mathbb{A}'$ (we assume that any two such sets $\mathbb{A}$ and $\mathbb{A}'$ are non-empty and disjoint for the rest of this section) is a triple $\mathcal{D} = (\Psi, \Psi', \beta)$, where $\Psi = \{\psi_i\}_{i\in I}$ and $\Psi' = \{\psi'_j\}_{j\in J}$ for index sets I and J, where β is a positive boolean formula with variables ranging over $\{x_i\}_{i\in I} \cup \{x'_j\}_{j\in J}$, each $\psi \in \Psi$ (resp. each $\psi' \in \Psi'$) is an EF formula that is interpreted on the first (resp. second) component, i.e. over $\mathbb{A}$ (resp. $\mathbb{A}'$).

Recall that such a decomposition has the property that for every pointed transition system $(\mathcal{T}, s)$ over $\mathbb{A}$ and every pointed transition system $(\mathcal{T}', s')$ over $\mathbb{A}'$ and every EF formula φ over $\mathbb{A} \cup \mathbb{A}$ we have $((\mathcal{T} \times \mathcal{T}'), (s, s')) \models \varphi$ if and only if $\beta[\mu]$ is true, where $\mu(x_i) = 1$ if and only if $(\mathcal{T}, s) \models \psi_i$ and where $\mu(x'_j) = 1$ if and only if $(\mathcal{T}', s') \models \psi'_j$. As expected, the *size* of such a decomposition is defined as $|\mathcal{D}| = \sum_{\psi \in \Psi} |\psi| + \sum_{\psi' \in \Psi'} |\psi'| + |\beta|$.

▶ **Theorem 5.** *The size of decompositions for* EF *(resp.* HM*) formulas in the sense of Theorem 1 cannot be bounded by an elementary function. More precisely, there is a family of* EF *(resp.* HM*) formulas* $\{\varphi_\ell \mid \ell \geq 1\}$ *where* φ_ℓ *is defined over some action labels* $\mathbb{A}_\ell \cup \mathbb{A}'_\ell$*, such that* $|\varphi_\ell| = \exp(\ell)$*, and such that for every elementary function* $f : \mathbb{N} \to \mathbb{N}$ *there is some* $h \in \mathbb{N}$ *such that every decomposition* $\mathcal{D}$ *for* φ_h *on the class of all asynchronous products of two transition systems over, respectively,* $\mathbb{A}_h$ *and* $\mathbb{A}'_h$ *satisfies* $|\mathcal{D}| > f(h)$*.*

4.1 Proof of Theorem 5

The proof idea for Theorem 5 for the case of EF-logic is as follows (we will remark how to adapt it for HM-logic later). We consider the sequence of pairs of BPAs $\{(\mathcal{P}_\ell, \mathcal{P}'_\ell)\}_{\ell \geq 1}$ defined in the previous section, where the set of states of $\mathcal{T}(\mathcal{P}_\ell)$ (resp. $\mathcal{T}(\mathcal{P}'_\ell)$) is Σ_ℓ^* (resp. $(\Sigma'_\ell)^*$). We will show that if a small (i.e. of elementary size) decomposition for EF-formulas exists in general, then there is a family of DFAs $\mathcal{A}_\ell$ of size elementary in ℓ with $L(\mathcal{A}_\ell) = L_{\ell,\ell}$ for each ℓ, clearly contradicting Lemma 3. To this end, we invoke the result from [2] about the sizes of automata expressing the sets of configurations of BPAs satisfying EF formulas combined with standard constructions from automatic structures.

We first recall the following proposition from [2] about the size of DFAs representing the set of configurations of BPAs satisfying EF formulas.

▶ Proposition 6 ([2]). Given an EF formula φ and a BPA $\mathcal{P} = (\Sigma, \mathbb{A}, \Delta)$, there exists a DFA $\mathcal{A}_\varphi$ of size double exponential in $|\mathcal{P}| + |\varphi|$ with $L(\mathcal{A}_\varphi) = [\![\varphi]\!]_{\mathcal{T}(\mathcal{P})}$, i.e. $\mathcal{A}_\varphi$ accepts the set of states u of $\mathcal{T}(\mathcal{P})$ with $(\mathcal{T}(\mathcal{P}), u) \models \varphi$.

Actually, in [2], the authors construct alternating finite automata with polynomially many states, which can be translated to DFAs of double exponential size (e.g. see [17]).

Define $\{\varphi_\ell \mid \ell \geq 1\}$ as $\varphi_\ell = \mathsf{xcount}_{(\ell,\ell)}$ over the action labels $\mathbb{A}_\ell = \mathbb{L}_\ell$ and $\mathbb{A}'_\ell = \mathbb{L}'_\ell$, where recall that $\mathbb{L}_\ell$ (resp. $\mathbb{L}'_\ell$) are the action labels of the BPA $\mathcal{P}_\ell$ (resp. $\mathcal{P}'_\ell$) defined in the previous section.

To prove Theorem 5, assume to the contrary that the there exist decompositions for EF formulas φ whose sizes can be bounded from above by an elementary function, say by $\mathsf{Tower}(r, |\varphi|)$ for some *fixed* $r \in \mathbb{N}$. Let $h \in \mathbb{N}$ be a sufficiently large number for the following arguments to work. Let us fix a smallest possible decomposition $\mathcal{D} = (\Psi, \Psi', \beta)$ for the EF formula $\varphi_h = \mathsf{xcount}_{(h,h)}$ over $\mathbb{L}_h \cup \mathbb{L}'_h$. Thus by assumption $|\mathcal{D}| \leq \mathsf{Tower}(r, |\varphi_h|)$. Let $\Psi = \{\psi_i\}_{i \in I}$ and $\Psi' = \{\psi'_j\}_{j \in J}$. Recall that each $\psi_i \in \Psi$ is an EF formula over $\mathbb{L}_h$, and each $\psi'_j \in \Psi'$ is an EF formula over $\mathbb{L}'_h$. Moreover β is a positive boolean formula over the variables $\{x_i\}_{i \in I} \cup \{x'_j\}_{j \in J}$ such that for every state $(u, u') \in (\Sigma_h)^* \times (\Sigma'_h)^*$ of $\mathcal{T}(\mathcal{P}_h) \times \mathcal{T}(\mathcal{P}'_h)$, it is the case that $(\mathcal{T}(\mathcal{P}_h) \times \mathcal{T}(\mathcal{P}'_h), (u, u')) \models \varphi_h$ if and only if $\beta[\mu]$ is true, where μ is the assignment to β where we have $\mu(x_i) = 1$ if and only if $(\mathcal{T}(\mathcal{P}_h), u) \models \psi_i$ and $\mu(x'_j) = 1$ if and only if $(\mathcal{T}(\mathcal{P}'_h), u') \models \psi'_j$.

Next, we will use Proposition 6 and the small decomposition given by the assumption to construct a DFA for the language $L_{h,h} = \{u \in \Sigma_h^* \mid \exists u \in (\Sigma'_h)^* : (\mathcal{T}(\mathcal{P}_h) \times \mathcal{T}(\mathcal{P}'_h), (u, u')) \models \varphi_h\}$ with less than $\mathsf{Tower}(h-1, h) + 1$ states, which will contradict Lemma 3. To do so, we first make the following simple observation that relates the decomposition $\mathcal{D}$ of φ_h and the formula φ_h itself.

Define the EF formula $\widetilde{\beta}$ over $\mathbb{L}_h \cup \mathbb{L}'_h$ to be obtained from the boolean formula β by replacing each variable x_i by ψ_i and each variable x'_j by ψ'_j. Then, since all formulas ψ_i and ψ'_j are also formulas over $\mathbb{L}_h \cup \mathbb{L}'_h$, the EF formula $\widetilde{\beta}$ is also a formula over $\mathbb{L}_h \cup \mathbb{L}'_h$. Moreover, it is easy to see that by assumption we have $[\![\varphi_h]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)} = [\![\widetilde{\beta}]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)}$. In fact, the latter immediately follows from the fact that

$$[\![\psi_i]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)} = [\![\psi_i]\!]_{\mathcal{T}(\mathcal{P}_h)} \times (\Sigma'_h)^*, \quad \text{and} \tag{1}$$

$$[\![\psi'_j]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)} = \Sigma_h^* \times [\![\psi'_j]\!]_{\mathcal{T}(\mathcal{P}'_h)} \tag{2}$$

which can easily be proven by induction on the structure of the formulas ψ_i and ψ'_j since no action labels of $\mathcal{P}_h$ (resp. $\mathcal{P}'_h$) occur in the action labels of ψ'_j (resp. ψ_i). Thus, the goal to obtain a contradiction will be to show that we can find a small DFA for

$$L_1(\widetilde{\beta}) = \{u \in \Sigma_h^* \mid \exists u' \in (\Sigma'_h)^* : (\mathcal{T}(\mathcal{P}_h) \times \mathcal{T}(\mathcal{P}'_h), (u, u')) \models \widetilde{\beta}\}.$$

Using Proposition 6 we obtain DFAs for $[\![\psi_i]\!]_{\mathcal{T}(\mathcal{P}_h)}$ (for each $i \in I$) and $[\![\psi'_j]\!]_{\mathcal{T}(\mathcal{P}'_h)}$ (for each $j \in J$) each of size double exponential in, respectively, $|\psi_i| + |\mathcal{P}_h|$ and $|\psi_j| + |\mathcal{P}'_h|$. To obtain a small DFA for $L_1(\widetilde{\beta})$ from these DFAs, we will now perform some simple constructions from automatic structures (e.g. see [16]). We first briefly recall the notion of (binary) automatic relations. Fix a nonempty finite alphabet Σ. A pair of words $(u, w) = (a_1 \cdots a_m, b_1 \cdots b_n) \in \Sigma^* \times \Sigma^*$ can be represented as a word $u \otimes w = c_1 \cdots c_k$ of length $k = \max(m, n)$ in the new alphabet $\Sigma_\perp \times \Sigma_\perp$, where $\Sigma_\perp = \Sigma \cup \{\perp\}$ with a "padding" symbol $\perp \notin \Sigma$, and where either $c_i = (a_i, b_i)$ if $i \leq m$ and $i \leq n$, where $c_i = (a_i, \perp)$ if $i \leq m, i > n$ or where $c_i = (\perp, b_i)$ otherwise. A (binary) relation $R \subseteq \Sigma^* \times \Sigma^*$ is said to be *automatic* if the language $\{u \otimes v \mid (u, v) \in R\} \subseteq (\Sigma_\perp \times \Sigma_\perp)^*$ can be accepted by a DFA (i.e. is regular). We also write $\pi_1(R)$ to be the projection of R to the first component, i.e., $\pi_1(R) = \{u \in \Sigma^* \mid \exists w : (u, w) \in R\}$. The following proposition is standard (e.g. see [16]):

▶ Proposition 7. Given two automatic relations R_1, R_2 accepted by DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively, we have (i) the relation $R_1 \cap R_2$ can be accepted by a DFA of size at most $|\mathcal{A}_1| \cdot |\mathcal{A}_2|$, (ii) the relation $R_1 \cup R_2$ can be accepted by a DFA of size at most $|\mathcal{A}_1| \cdot |\mathcal{A}_2|$, and (iii) the language $\pi_1(R_1) \subseteq \Sigma^*$ can be accepted by a DFA of size $2^{O(|\mathcal{A}_1|)}$.

Observe now that $[\![\psi_i]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)}$ (for each $i \in I$) and $[\![\psi'_j]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)}$ (for each $j \in J$) is an automatic relation over the alphabet $\Sigma = \Sigma_h \cup \Sigma'_h$ that can be accepted by DFAs of size double exponential in, respectively, $|\psi_i| + |\mathcal{P}_h| + |\mathcal{P}'_h|$ and $|\psi'_j| + |\mathcal{P}_h| + |\mathcal{P}'_h|$ by Proposition 6. The construction of a small DFA $\mathcal{A}$ for the language $L_1(\widetilde{\beta})$ can be done in a bottom-up fashion with respect to $\widetilde{\beta}$ using Proposition 7 by firstly taking unions and intersections from the DFAs recognizing $[\![\psi_i]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)}$ (for each $i \in I$) and $[\![\psi'_j]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)}$ (for each $j \in J$), and at the end projecting to the first component. All in all, there are constants c_1, c_2 with $c_1 < c_2$ (both independent of h) such that

$$|\mathcal{A}| \leq \mathsf{Tower}(c_1, |\varphi_h| + |\mathcal{P}_h| + |\mathcal{P}'_h|) \leq \mathsf{Tower}(c_2, h). \tag{3}$$

The latter inequality follows from the fact that $|\mathcal{P}_h| + |\mathcal{P}'_h| = \mathsf{poly}(h)$ and $|\varphi_h| = \exp(h)$. On the other hand, due to $[\![\varphi_h]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)} = [\![\widetilde{\beta}]\!]_{\mathcal{T}(\mathcal{P}_h)\times\mathcal{T}(\mathcal{P}'_h)}$ and Lemma 3, we must have

$$|\mathcal{A}| \geq \mathsf{Tower}(h - 1, h) + 1. \tag{4}$$

It is clear that if we choose h sufficiently large, then inequalities (3) and (4) cannot hold at the same time, a contradiction.

Remark. The proof above can be easily adapted to the case of HM-logic by taking prefix-recognizable systems and the HM formulas of the form $\widetilde{\mathsf{xcount}_{(\ell,\ell)}}$ defined in the remark given at the end of previous section instead of BPAs and EF formulas of the form $\mathsf{xcount}_{(\ell,\ell)}$.

4.2 Restricting to finite transition systems

Theorem 5 gives a nonelementary lower bound for decompositions over asynchronous products of two general transition systems. This still leaves the possibility that better upper bounds might be possible when we consider only asynchronous products of *finite* transition systems, i.e., the version of Theorem 1 when transition systems under consideration are finite. The following theorem shows that this is not the case.

▶ **Theorem 8.** *The size of decompositions for* EF *(resp.* HM*) formulas in the sense of Theorem 1 cannot be bounded by an elementary function when restricted to the class of finite transition systems.*

Roughly speaking, this theorem can be proven by combining Theorem 5 and the fact that HM and EF logics satisfy "finite model property with respect to a finite set of formulas": a logic $\mathcal{L}$ is said to satisfy the *finite model property with respect to a finite set of formulas* whenever, for every finite set Ξ of $\mathcal{L}$-formulas and every pointed transition system $(\mathcal{T}, s)$ there exists a *finite* pointed transition system $(\mathcal{T}_\Xi, s_\Xi)$ such that for all $\psi \in \Xi$ we have $(\mathcal{T}, s) \models \psi$ if and only if $(\mathcal{T}_\Xi, s_\Xi) \models \psi$.

It simple to check that when restricted to logics that are closed under boolean operations the finite model property with respect to a finite set of formulas is equivalent to the the finite model property (for single formulas). To prove Theorem 8, the following technical lemma suffices.

▶ **Lemma 9.** *Let φ be an* HM *(resp.* EF*) formula over the action labels $\mathbb{A} \cup \mathbb{A}'$ for nonempty disjoint sets $\mathbb{A}$ and $\mathbb{A}'$. Then, every decomposition of φ over all asynchronous products of two finite systems over $\mathbb{A}$ and $\mathbb{A}$, respectively, is also a decomposition of φ over all asynchronous products of two general transition systems over $\mathbb{A}$ and $\mathbb{A}'$, respectively.*

Observe that Theorem 5 and the above lemma immediately imply Theorem 8. We shall give the proof of Lemma 9 for EF-logic. In fact, HM-logic can be dealt with completely analogously. We note that EF (analogously HM) has the finite model property with respect to a finite set of formulas owing to the same property for *Propositional Dynamic Logic* (of which EF-logic is a sublogic). The latter can be proven e.g. via filtration (e.g. see [8]).

Let us now proceed to the proof of Lemma 9. Let fix an arbitrary EF-formula φ. Let $\mathcal{C} = (\Phi, \Phi', \alpha)$ be a decomposition of φ over the class asynchronous products of two *finite* transition systems over $\mathbb{A}$ and $\mathbb{A}'$, respectively. Let us assume $\Phi = \{\varphi_k\}_{k\in K}$ and $\Phi' = \{\varphi'_m\}_{m\in M}$ for index sets K and M. We assume here that α is a boolean formula over the variables $\{x_k\}_{k\in K}$ and $\{x'_m\}_{m\in M}$. It is important to note here that we may only assume here that $\mathcal{C}$ is a decomposition for φ on the class of asynchronous products of two *finite* transition systems over $\mathbb{A}$ and $\mathbb{A}'$, respectively.

In addition, we apply Theorem 1 to obtain a decomposition $\mathcal{D} = (\Psi, \Psi', \beta)$ of φ over the class of asynchronous products of two *general* transition systems, where $\Psi = \{\psi_i\}_{i\in I}$ and $\Psi' = \{\psi_j\}_{j\in J}$. We assume that β is a boolean formula over the variables $\{y_i\}_{i\in I}$ and $\{y'_j\}_{j\in J}$.

Let us define the finite set of formulas $\Xi = \Phi \cup \Psi$ over $\mathbb{A}$ and $\Xi' = \Phi' \cup \Psi'$ over $\mathbb{A}'$. Let us fix an arbitrary pointed transition system $(\mathcal{T}, s)$ over $\mathbb{A}$ and an arbitrary transition system $(\mathcal{T}', s')$ over $\mathbb{A}'$. Let us moreover fix some *finite* pointed transition systems $(\mathcal{T}_\Xi, s_\Xi)$ over $\mathbb{A}$ and $(\mathcal{T}'_{\Xi'}, s'_{\Xi'})$ over $\mathbb{A}'$ witnessing the finite model property with respect to Ξ and Ξ', respectively. To prove to lemma we will show that already $\mathcal{C}$ is a decomposition, so we will show $(\mathcal{T} \times \mathcal{T}', (s, s')) \models \varphi$ if and only if α is true, if $x_k = ((\mathcal{T}, s) \models \varphi_k)$ and

$x'_m = ((\mathcal{T}', s') \models \varphi'_m)$. The latter follows from the following equivalences:

$$\begin{aligned}
(\mathcal{T} \times \mathcal{T}', (s, s')) \models \varphi \quad &\Leftrightarrow \quad \beta \text{ is true if } y_i = ((\mathcal{T}, s) \models \psi_i) \text{ and } y'_j = ((\mathcal{T}', s') \models \psi'_j).\\
&\Leftrightarrow \quad \beta \text{ is true if } y_i = ((\mathcal{T}_\Xi, s_\Xi) \models \psi_i) \text{ and } y'_j = ((\mathcal{T}'_{\Xi'}, s'_{\Xi'}) \models \psi'_j).\\
&\Leftrightarrow \quad (\mathcal{T}_\Xi \times \mathcal{T}'_{\Xi'}, (s_\Xi, s'_{\Xi'})) \models \varphi.\\
&\Leftrightarrow \quad \alpha \text{ is true if } x_k = ((\mathcal{T}_\Xi, s_\Xi) \models \varphi_k) \text{ and } x'_m = ((\mathcal{T}'_{\Xi'}, s'_{\Xi'}) \models \varphi'_m).\\
&\Leftrightarrow \quad \alpha \text{ is true if } x_k = ((\mathcal{T}, s) \models \varphi_k) \text{ and } x'_m = ((\mathcal{T}', s') \models \varphi'_m).
\end{aligned}$$

Hence, $\mathcal{C}$ is a decomposition over all (both finite and infinite) transition systems, completing the proof of Lemma 9.

Acknowledgements: Anthony Lin thanks EPSRC (EP/H026878/1) for their support.

References

1. P. Blackburn, M. de Rijke, Y. Venema. *Modal Logic*. Cambridge University Press, 2001.
2. A. Bouajjani, J. Esparza, O. Maler. Reachability Analysis of Pushdown Automata: Applications to Model-Checking. In *CONCUR'97*, p. 135–150.
3. T. Cachat, I. Walukiewicz. The Complexity of Games on Higher Order Pushdown Automata. In *arxiv* http://arxiv.org/abs/0705.0262
4. D. Caucal. On the Regular Structure of Prefix Rewriting In *CAAP'90*, p. 87–102.
5. D. Caucal. On infinite transition graphs having a decidable monadic theory. In *ICALP'96*, p. 194–205.
6. A. Dawar, M. Grohe, S. Kreutzer, N. Schweikardt. Model Theory Makes Formulas Large. In *ICALP'07*, p. 913–924.
7. I. Felscher, W. Thomas. Compositionality and Reachability with Conditions on Path Lengths. *Int. J. Found. Comput. Sci.* 20(5): 851–868 (2009)
8. D. Harel, D.Kozen, J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.
9. O. Kupferman, N. Piterman, M. Y. Vardi. Model checking linear properties of prefix-recognizable systems. In *CAV'02*, p. 371–385.
10. L. Libkin. *Elements of Finite Model Theory*. Springer, 2004.
11. S. Lu, S. Park, E. Seo, Y. Zhou. Learning from mistakes: a comprehensive study on real world concurrency bug characteristics. In *ASPLOS'08*, p. 329–339.
12. J. A. Makowsky. Algorithmic uses of the Feferman-Vaught Theorem. *Ann. Pure Appl. Logic* 126(1–3): 159–213 (2004)
13. R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, TU München, 1998.
14. S. Qadeer, J. Rehof. Context-Bounded Model Checking of Concurrent Software. In *TACAS'05*, p. 93–107.
15. A. Rabinovich. On compositionality and its limitations. *ACM TOCL* 8(1): (2007)
16. A. W. To. Model Checking Infinite-State Systems: Generic and Specific Approaches. PhD thesis, University of Edinburgh, 2010.
17. M. Vardi. An Automata-Theoretic Approach to Linear Temporal Logic In *Banff Higher Order Workshop '95*, p. 238-266.
18. S. Wöhrle, W. Thomas. Model Checking Synchronized Products of Infinite Transition Systems. *LMCS* 3(4): (2007)
19. I. Walukiewicz. Difficult Configurations - On the Complexity of LTrL In *ICALP '98*, p. 140–151.
20. I. Walukiewicz. Model Checking CTL Properties of Pushdown Systems. In *FSTTCS'00*, p. 127–138.

Conflict-free Chromatic Art Gallery Coverage*

Andreas Bärtschi[1] and Subhash Suri[2]

1 **Institute of Theoretical Computer Science, ETH Zürich**
8092 Zürich, Switzerland (andrbaer@student.ethz.ch)
2 **Department of Computer Science, University of California**
Santa Barbara, CA 93106, USA (suri@cs.ucsb.edu)

Abstract

We consider a *chromatic* variant of the art gallery problem, where each guard is assigned one of k distinct colors. A placement of such colored guards is *conflict-free* if each point of the polygon is seen by some guard whose color appears exactly once among the guards visible to that point. What is the smallest number $k(n)$ of colors that ensure a conflict-free covering of all n-vertex polygons? We call this the *conflict-free chromatic art gallery problem*. The problem is motivated by applications in distributed robotics and wireless sensor networks where colors indicate the wireless frequencies assigned to a set of covering "landmarks" in the environment so that a mobile robot can always communicate with at least one landmark in its line-of-sight range without interference. Our main result shows that $k(n)$ is $O(\log n)$ for orthogonal and for monotone polygons, and $O(\log^2 n)$ for arbitrary simple polygons. By contrast, if *all* guards visible from each point must have distinct colors, then $k(n)$ is $\Omega(n)$ for arbitrary simple polygons and $\Omega(\sqrt{n})$ for orthogonal polygons, as shown by Erickson and LaValle [3].

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases art gallery problem, conflict-free coloring, visibility

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.160

1 Introduction

The Art Gallery Theorem is a classical result in computational geometry, first posed by Klee and proved by Chvátal [2], which says that $\lfloor n/3 \rfloor$ (point) guards are always sufficient, and sometimes necessary, to cover a simply-connected n-vertex polygon. In the last 30 years, many extensions, variations, and generalizations involving different types of guards, polygons, and visibility constraints have been investigated. (See [6] and [8], for instance.)

Besides their mathematical elegance and appeal, the interest in art gallery problems is also spurred by applications in distributed surveillance, monitoring, and robotics. In many of these applications, the "guards" are "landmarks" deployed in an environment to help provide navigation and localization service to mobile robots. The mobile device communicates with these landmarks through wireless, or other "line-of-sight" signaling mechanisms. In order for the signaling mechanism to work correctly, the different landmarks visible to the robot at any position must operate on different frequency—the robot is unable to receive the signal if multiple landmarks in its range are transmitting at the same frequency. This motivates a "chromatic" version of the art gallery theorem, where the goal is not to optimize the *number of guards*, but rather the *number of distinct colors* needed to distinguish the guards.

* Andreas Bärtschi's research was partially supported by a scholarship from the Student Exchange Office of ETH Zürich. Subhash Suri's research was supported in part by the NSF grant IIS-0904501.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 160–171

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

Problem Motivation and the Results

Radio transceivers are cheap but tuning them to many different frequencies requires costly hardware. If the polygons can be covered by guards of very few *distinct colors* (frequencies), then it would enable inexpensive robot localization and navigation. This was the motivation behind the work of Erickson and LaValle [4] who sought to guard the polygon so that each point of the polygon is seen by guards of distinct colors only—that is, the robot located anywhere in the polygon is able to communicate without interference with *any* of the guards in its line-of-sight. Surprisingly, Erickson and LaValle discovered that this *strong chromatic* condition does not lead to much savings in the number of colors: there are simple polygons that require $\Omega(n)$ colors, and even monotone orthogonal polygons require $\Omega(\sqrt{n})$ colors [3].

Motivated by this negative result, we consider a weaker chromatic condition, which is sufficient for the original robotics application of interference-free communication with a guard at all locations. Specifically, we call a placement of colored guards *conflict-free* if each point of the polygon is seen by some guard whose color appears exactly once among the guards visible to that point. Thus, for any placement of the robot in the polygon, there is at least one guard that can communicate with the robot without interference. We want to determine the smallest number $k(n)$ of colors that ensure a conflict-free coloring of some guard set in all n-vertex polygons. We call this the *conflict-free chromatic art gallery problem*.

The main result of our paper is to prove that $k(n)$ is $O(\log n)$ for orthogonal and for monotone polygons, and $k(n) = O(\log^2 n)$ for arbitrary simple polygons. Thus, not only does the conflict-free coloring yield significantly smaller bounds for distinct colors, it also fulfills the hopeful vision of robotics application that a few colors suffice.

Related Work and Hypergraph Coloring

The chromatic art gallery problem is related to hypergraph coloring, where one must assign colors to the vertices of a hypergraph $H = (V, \mathcal{E})$, so that its edges, which are subsets of vertices, are appropriately colored. In the most basic form, called the proper coloring, every edge e with at least two vertices must be non-monochromatic; that is, there must be two vertices $x, y \in e$ whose colors are distinct. In the conflict-free coloring of H, every edge e must have a vertex that is uniquely colored among the vertices in e. Smorodinsky [9, 11] considers several simple geometric hypergraphs, such as those induced by disks or rectangles. For instance, the rectangle hypergraph has a finite set of axis-aligned rectangles, and each maximal subset of rectangles with a common intersection forms an hyperedge. For these hypergraphs, it is known that the conflict-free chromatic number is $\Omega(\log n)$ and $O(\log n)$ [7, 10].

To see the connection between chromatic art gallery and the hypergraph coloring, consider a guard set S, and let $\mathcal{R}$ be the set of the guards' visibility regions in the polygon. Then we have a hypergraph $H = (V, \mathcal{E})$, whose vertices correspond to S and in which a subset $S_e \subseteq S$ corresponds to an edge if there is a point p_e in the polygon contained exactly in the visibility regions of the guards in S_e and no others. A *conflict-free hypergraph coloring* of H is easily seen to be also a conflict-free coloring of the guard set S. Of course, in the chromatic art gallery, we need to simultaneously choose the guard set and color it, so it does not quite reduce to the hypergraph coloring. Even if we were to consider a fixed guard set, the visibility regions are not as well-behaved as disks or rectangles, and no non-trivial bound is known for their conflict-free chromatic number.

The previous result that is most directly relevant to our work is the mentioned version of the chromatic art gallery, with a stronger chromatic condition on the guard's coloring. This original version relates to a *strong hypergraph coloring* of the corresponding hypergraph H.

Organization

Section 2 introduces some basic definitions and concepts. In Section 3, we prove the $O(\log n)$ bound for the conflict-free coloring of orthogonal polygons, and the general proof strategy that is used later for simple polygons as well. In Section 4, we prove the $O(\log n)$ bound for monotone polygons, which is the key to establishing the $O(\log^2 n)$ upper bound for general polygons in Section 5.

2 The conflict-free chromatic art gallery problem

Let P be a simple polygon, whose boundary we denote as $\partial P \subset P$. We say that two points $p, q \in P$ are *visible* to each other if the line segment $\overline{pq}$ is a subset of P. The *visibility region* of a point p is defined as $V(p) := \{q \in P \mid q \text{ is visible from } p\}$. A finite point set $S \subset P$ is called a *guard set* if $\bigcup_{p \in S} V(p) = P$ and we call the points in S *guards*. A coloring $c\colon S \to \{1, \ldots, k\}$ of the guards with k colors is called *conflict-free* if each point $p \in P$ is seen by a guard whose color appears exactly once among all guards that see p. Let $k_{cf}(S)$ be the minimum number of colors required to color a guard set S conflict-free and let $\mathcal{S}(P)$ be the set of all guard sets of P. Then the *conflict-free chromatic guard number* of a polygon P is defined as $\chi(P) := \min_{S \in \mathcal{S}(P)} k_{cf}(S)$. We want to determine the smallest number $k(n)$ such that for all n-vertex polygons P_n we have $\chi(P_n) \leq k(n)$.

The classical art gallery theorem says that $\lfloor n/3 \rfloor$ guards are both necessary and sufficient for covering a n-vertex polygon, but the number of colors needed to ensure conflict-free covering may be significantly smaller. For instance, the construction that forces $\lfloor n/3 \rfloor$ guards (Fig. 1) only requires *two colors*. A polygon is called *orthogonal*, if its edges meet at right angles. A polygon P is called *monotone with respect to a line* ℓ if every line orthogonal to ℓ intersects the boundary of P at most twice. P is called *x-monotone* (y-monotone) if P is monotone with respect to the x-axis (respectively the y-axis).

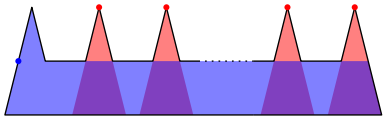


Figure 1 This polygon requires $\lfloor n/3 \rfloor$ guards but its conflict-free chromatic guard number is just 2.

The following concept of *independence* is central to our proofs, and forms a basis for coloring by partitioning into independent subpolygons.

▶ **Definition 1** (Independence). Let P be a polygon. We call two subpolygons P_1 and P_2 of P *independent* if there are no points $p_1 \in P_1$ and $p_2 \in P_2$ that are mutually visible.

▶ **Lemma 2.** *Let $\{A_1, \ldots, A_m\}$ be a partition of the polygon P into m families of pairwise independent subpolygons. That is, each $A_i = \{P_{i1}, \ldots, P_{ik_i}\}$ is a collection of subpolygons that are pairwise independent and all the subpolygons in the m families form a partition of P. Then we have* $\chi(P) \leq \sum_{i=1}^{m} \max_{P_{ij} \in A_i} \{\chi(P_{ij})\}$.

Proof. Let $\{C_1, \ldots, C_m\}$ be m disjoint color sets, where $|C_i| = \max_{P_{ij} \in A_i} \{\chi(P_{ij})\}$. Then we can guard every subpolygon $P_{ij} \in A_i$ conflict-free in itself with guards that get colors from C_i, giving a total number of $|C_1| + \ldots + |C_m|$ colors. We claim that this coloring ensures that every point $p \in P$ sees a guard of unique color among all guards that see p. To prove this claim, without loss of generality, suppose that p is contained in a subpolygon P_{ij_1} of A_i and s_1 is its guard of unique color in P_{ij_1}. Any other guard s_2 in P that has the same color as s_1 must lie in a subpolygon $P_{ij_2} \neq P_{ij_1}$, which is contained in A_i and hence independent of P_{ij_1}. Thus s_2 does not see p, and s_1 is not only a guard of unique color among all guards in P_{ij_1}, but among all guards in P. Thus, we have found a conflict-free covering with $|C_1| + \ldots + |C_m|$ colors, which completes the proof. ◀

Lemma 2 naturally suggests a divide-and-conquer strategy: we partition the polygon into four sets of subpolygons and then conquer each set by recursively splitting the regions into sets of independent regions and applying Lemma 2.

▶ Remark. We only require the interiors of subpolygons P_1 and P_2 to be independent, and allow mutual visibility among their boundary points as long as these points *also belong* to the boundary of another subpolygon that is responsible for their conflict-free covering. In particular, for a line segment e contained in two boundaries ∂P_1 and ∂P_2, we will explicitly mention whether P_1 or P_2 is "responsible" for guarding e.

3 Orthogonal Polygons

Our basic strategy is to partition the orthogonal polygon P into four types of monotone orthogonal subpolygons. These subpolygons have a boundary consisting of a single base edge and another subchain that is either x-monotone or y-monotone. The chain can be either above the base edge or below in the former case, and to the left or to the right in the latter case. We use mnemonic identifiers U (up), D (down), L (left) and R (right) to refer to these four types. When we show all or parts of the partition, we display these types with the colors red, green, black and blue, always using the following consistent mapping U → red, D → green, L → black and R → blue.

The partitioning process

Given a polygon P we construct a partition by iteratively adding monotone subpolygons. In each *odd-numbered* step we add subpolygons of Type U and D, and in each *even-numbered* step we add subpolygons of Type L and R. Figures 2 and 3 illustrate the construction.

Step 1 Let e be the lowest horizontal edge of P's boundary. Let Q be the set of all points $q \in P$ which are vertically visible from e and lie on or above e. Q is the first subpolygon in our partitioning, and it is of type U. Because P is a simply-connected region, with no holes, it is easy to see that Q splits it in parts that lie entirely to its left or entirely to its right, and each part R shares exactly one edge with Q, which is a vertical line segment.

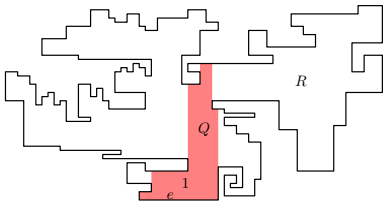


Figure 2 The first step of the partitioning process.

Step 2 The line segments on the boundary of Q become the base edges for new subpolygons of Type L and R, which are defined analogously as the first subpolygon, with vertical visibility replaced by horizontal visibility. We note that the remaining regions lie entirely above or below a subpolygon of type L or R and share exactly one horizontal line segment with these subpolygons, but not with the first subpolygon Q.

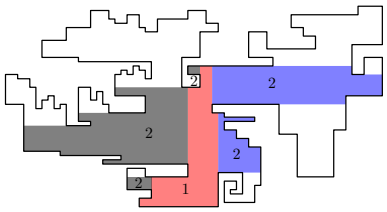


Figure 3 The second step of the partitioning process.

Step 3 The horizontal line segments from Step 2 in turn generate subpolygons of Type U and D.

We repeat steps 2 and 3 until we have a complete partition. In each odd-numbered step we construct red (U) and green (D) polygons and in each even-numbered step black (L) and blue (R) subpolygons.

▶ **Lemma 3.** *The partitioning process terminates within $n + 2$ steps.*

Proof. In each step at least one subpolygon is added to the partition. Such a subpolygon touches at least one edge $e = \{u, v\}$ previously not touched. In at most two additional steps, both the endpoints of e, u and v, become completely surrounded by subpolygons of the partition. The polygon is completely covered if all vertices are surrounded, hence the partitioning process ends after at most $n + 2$ steps. ◀

The schematic tree

The recursive partitioning generates four *families* of polygons: up-polygons A_U, down-polygons A_D, left-polygons A_L, and right-polygons A_R. Ideally, we would like to invoke Lemma 2 on this partition partitioned $\{A_U, A_D, A_L, A_R\}$. Unfortunately the subpolygons in each family are *not independent*, see Fig. 4 for an example. We, therefore, introduce a condition that allows us to subdivide the group A_U into sets of independent subpolygons. In the following, we focus exclusively on the red (up) polygon group; the other three groups are handled in the same way.

We first introduce a schematic tree that is a convenient graphical representation of the polygon partition we have. This graph is a 4-colored directed graph, where each vertex represents a subpolygon of the partition of the same color. There exists a directed edge from a subpolygon P_i to a subpolygon P_j if and only if P_j has been constructed over a line segment e that is part of P_i's boundary. As mentioned earlier, we consider e to be part of P_i but not of P_j. Since P has no holes, T contains no cycle and is a tree. The first constructed

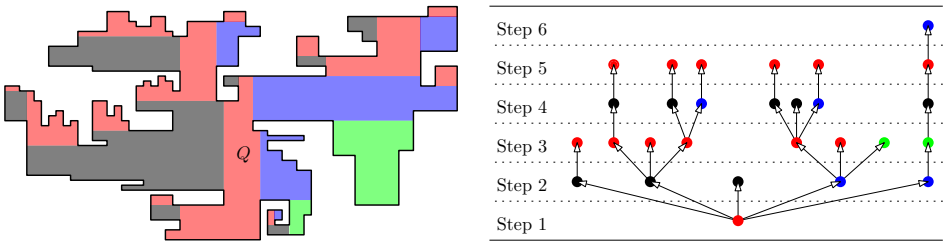


Figure 4 The complete partition and the corresponding schematic tree.

subpolygon Q has no incoming edge, and it represents the root of our tree. (The base edge of Q is considered to be a part of this subpolygon.) Since all other vertices have indegree 1, T is a rooted directed tree and any subpolygon constructed in Step k has depth $k - 1$ in T. Hence all red and green vertices have even height and all vertices of color blue or black have odd height. Therefore every directed path in T alternates between vertices of red or green color and vertices of blue or black color.

▶ Remark. Let $p_i \in P_i$ and $p_j \in P_j$ be two points of two subpolygons of the partition. Then the shortest path between p_i and p_j in P goes through a subpolygon P_k if and only if P_k lies on the shortest path between P_i and P_j in T.

▶ **Lemma 4.** *Let P be a polygon with the given partition and the schematic tree T. Let P_i and P_j be two arbitrary subpolygons of type U. Then, either (i) P_i and P_j are independent, or (ii) there exists a red-black-alternating (or a red-blue-alternating) directed path in T between P_i and P_j.*

Proof. Suppose P_i and P_j are not independent, then there exist points $p_i \in P_i$ and $p_j \in P_j$ that are mutually visible. The shortest path in P between p_i and p_j, therefore, must be a

line segment. The way we included the base edges to be part of just one subpolygon excludes the possibility of the line segment being horizontal or vertical. Without loss of generality, let us assume that the line segment is directed up and to the left, with p_i at the bottom-right, and p_j at the top-left. Since P_i is a U polygon, the visibility ray $\overrightarrow{p_ip_j}$ can only leave it through its left boundary, and therefore it must enter a type L subpolygon. Next, by the upward direction of $\overrightarrow{p_ip_j}$, it can leave this L subpolygon only through a top boundary edge, which forces it to enter a U subpolygon. This process repeats until we reach P_j, showing that the sequence of polygons traversed by the shortest path from p_i to p_j is an alternating U-L sequence, which corresponds to a red-black-alternating path in T. ◀

Conquering red-black-alternating trees: Staircase and recursion

Deriving a bound on the conflict-free chromatic guard number for family A_U directly seems difficult, because of inter-dependence of the subpolygons within the family. Instead, we use the property of Lemma 4 to look at that portion of A_U that forms a *red-black-alternating tree*. That is, consider the union of the subpolygons that corresponds to a red-black alternating tree in T. Suppose P_n is such an n-vertex orthogonal polygon, namely, whose partition is a red-black alternating tree. We will cover a part of P_n with a *staircase polygon* in such a way that all other relevant parts (containing red subpolygons) are independent and proceed recursively for all of them.

Recall that a *staircase (orthogonal) polygon* is an orthogonal polygon whose boundary can be split into two subchains with alternating convex and reflex interior vertices, with the two endpoints being convex. A staircase polygon in which one of the subchains has only one interior vertex is called a *convex fan*. Convex fans are star-shaped and can clearly be guarded with one guard (and one color).

▶ **Lemma 5.** *The conflict-free chromatic number for a staircase polygon P is at most* 3.

Proof. Consider the following placement of colored guards in a staircase polygon: Starting from the top, we place a guard s_1 on the first convex vertex of the lower subchain. Then we iteratively place a guard s_{i+1} on the lowest convex vertex visible from s_i, alternating between the two subchains until the staircase polygon is covered. To each guard s_i we assign the color in $\{1, 2, 3\}$ with the same residue class as i modulo 3. One can check that the coloring is conflict-free, and a complete proof can be found in [4]. ◀

Let $f(n)$ denote the smallest number of colors that ensure a conflict-free covering of all type U subpolygons in any orthogonal P_n corresponding to a red-black-alternating tree. In other words, for every P_n there is a guard set $S \subset P_n$ that can be colored with $f(n)$ colors such that each point of a type U subpolygon is seen by some guard whose color appears exactly once among the guards visible to that point. In the following we give a placement of colored guards, which shows that $f(n)$ is $O(\log n)$.

Since P_n consists of type U and type L subpolygons, it "grows to the left". Therefore we will cover P_n with staircases ascending to the left in a natural way: Let e be a horizontal edge with two reflex vertices. We call the horizontal line through e a *decision line*, see Figure 5. A decision line splits P_n in a lower part and two or more independent upper parts, of which at most one upper part contains more than $\lfloor n/2 \rfloor$ vertices. Starting from the lowest and rightmost vertex of P_n we construct a staircase ascending to the left, which at every decision line follows the upper part with the most vertices. We guard this staircase with colors $\{1, 2, 3\}$. Furthermore at every intersection of the staircase's lower subchain with a base edge of a type U subpolygon, we insert a convex fan that is oriented to the left and to

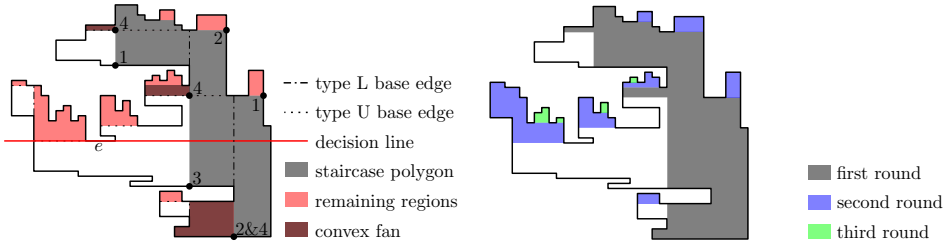


Figure 5 The first staircase subpolygon and covering with staircases and convex fans.

the top. These convex fans are bounded from the right by the staircase polygon and hence independent. We guard every convex fan with a guard of color 4 placed on the intersection. By iteratively adding staircases together with convex fans we can prove an upper bound on $f(n)$:

▶ **Lemma 6.** *Suppose P_n is an orthogonal polygon with a partition that has a red-black-alternating schematic tree. Then a conflict-free coloring of all the red regions of P_n needs at most $4 \log n$ colors. The same bound also holds for a red-blue-alternating schematic tree.*

Proof. We cover a part of the type U subpolygons with a staircase and convex fans as described. The remaining regions of the type U subpolygons are parts of smaller red-black-alternating trees. These smaller trees are all bounded from below by a decision line and from above and from the side by P_n's boundary, hence they are independent. Furthermore all of the smaller trees contain at most $\lfloor n/2 \rfloor$ of P_n's vertices because during the construction we choose at every decision line the upper part with the most remaining vertices.

Thus, the chromatic number follows the recurrence $f(n) \leq f(n/2) + 4$, which yields $f(n) \leq 4 \log n$. The same holds also for the red-blue-alternating trees by symmetry. ◀

A logarithmic upper bound for orthogonal polygons

We will show how one can cover *all* type U subpolygons in an arbitrary orthogonal polygon P_n with $O(\log n)$ colors. Let T be the schematic tree of the partition and let A and B be two disjoint color sets of size $f(n)$. We use the A and B to iteratively cover red-black-alternating and red-blue-alternating subtrees of T. In each step we must ensure that the subtrees of the same type are independent so that we can use the same colors for all of the subtrees:

Step 1 Take a not yet covered subpolygon P_s corresponding to a vertex v_s of minimal depth in T. Let T_s denote the inclusion-maximum red-black-alternating subtree rooted at v_s. By Lemma 6 we can guard all type U subpolygons corresponding to red vertices in the tree T_s with A.

Step 2 For every type U subpolygon in T_s (which now are all guarded) check whether it has red grandchildren in T that are not yet guarded (and thus must be connected through a blue vertex). These grandchildren are pairwise independent by Lemma 4, hence for each grandchild v it is possible to cover the inclusion-maximum red-blue-alternating subtree rooted at v with guards colored with colors in B conflict-free by Lemma 6. We have no conflicts with the type U subpolygons covered before since A and B are disjoint.

Step 3 As in Step 2, cover the independent inclusion-maximum red-black-alternating subtrees rooted at not yet covered red grandchildren of type U subpolygons in one of the red-blue-alternating subtrees. We use the color set A, which gives no conflicts with the guards in the red-blue-alternating subtrees, since they have colors from B. Furthermore we have also no conflicts with the guards in a previous red-black-alternating subtree by

Lemma 4, since the shortest path must go through the root of a red-blue-alternating subtree and hence through both a type L and a type R subpolygon.

Step 4 Repeat Step 2 and Step 3 as long as there are grandchildren. Otherwise we either have covered all type U subpolygons, or there remain type U subpolygons connected through a green vertex, which are thus independent by Lemma 4. In that case we start over with Step 1.

In this way, we get a conflict-free covering of all type U subpolygons in P_n with at most $|A| + |B| = 2f(n) = O(\log n)$ colors. We can apply the same procedure to type D, L and R subpolygons in alternating trees, since these cases are axis symmetric or rotationally symmetric. For each type we use two new color sets of size $f(n)$, which yields a conflict-free coloring of all subpolygons of the partition of an orthogonal polygon, where we use at most $8f(n) = O(\log n)$ colors in total. We have established the main result of this section.

▶ **Theorem 7.** *The conflict-free chromatic guard number for orthogonal polygons on n vertices is $k(n) = O(\log n)$.*

4 Monotone Polygons: a Step Towards Simple Polygons

The recursive partitioning technique of the previous section will form the basis for our proof of the general (non-orthogonal) polygons as well. However, the more complex visibility structure of non-orthogonal polygons forces us first to establish an intermediate result for *monotone* polygons. Specifically, our proof structure works by partitioning the polygon into families of simpler *staircase-shaped* subpolygons. In the orthogonal case, staircase polygons are easily covered using 3 colors (Lemma 5), but non-orthogonal staircases appear to be more complicated. The main result of this section is to show that this basic building block has conflict-free chromatic guard number $O(\log n)$. We then use this result to show that arbitrary simple polygons have conflict-free chromatic guard number $O(\log^2 n)$. A second (albeit minor) is that a naive recursive partitioning using x-aligned and y-aligned visibility may not even terminate in general polygons, and so we appropriately modify the partitioning to ensure finite termination. In the following, we assume without loss of generality that our polygon is x-monotone.

Monotone polygons

The monotone polygons are easily reduced to a collection of *independent* monotone polygons with a specialized structure, where one of the chains is either a line segment or a *concave chain*. Specifically, given an x-monotone polygon, consider the shortest path between the leftmost and the rightmost vertices. This path splits the polygon into a family of x-monotone pieces, with pieces of the shortest path forming one of their chains. In addition, all the subpolygons lying *below* the shortest paths are mutually independent, as are those lying above the path. Due to lack of space, the proof of the following lemma is omitted from this extended abstract.

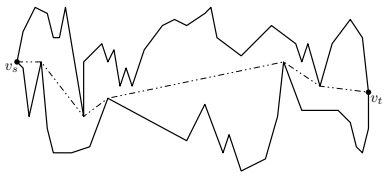


Figure 6 Partitioning a monotone polygon into independent monotone subpolygons over concave subchains.

▶ **Lemma 8.** *The conflict-free chromatic guard number for monotone polygons is at most twice the conflict-free chromatic guard number for monotone polygons over a concave chain.*

In the following, we show that monotone polygons over a concave chain have conflict-free chromatic guard number $O(\log n)$. The basic units of interest, however, turn out to be

monotone polygons over an edge or over a *convex chain*. The following subsection handles their coloring, which in turn forms the basis for coloring of monotone polygons over concave chains.

Monotone polygons over a convex subchain

Let P_n be a monotone polygon over a single horizontal edge. Let $g(n)$ denote the smallest number of colors that ensure a conflict-free covering for any such P_n. Similar to our method for constructing staircases in orthogonal polygons, we consider decision lines through either horizontal edges with adjacent reflex vertices or through a vertex for which both of its neighbors have a higher y-coordinate. A decision line splits P_n in a lower part and two or more independent upper parts, of which at most one part contains more than $\lfloor n/2 \rfloor$ vertices. Then we construct a subpolygon that contains the base edge of P_n and at each decision line follows the part with the most remaining vertices, see the left picture in Figure 7. This subpolygon is *star-shaped* and can thus be guarded with a single guard. The remaining regions are mutually independent, x-monotone over a horizontal edge and contain at most $\lfloor n/2 \rfloor$ of P_n's vertices. We get the recurrence $g(n) \leq g(n/2) + 1$, which yields $g(n) \leq \log n$.

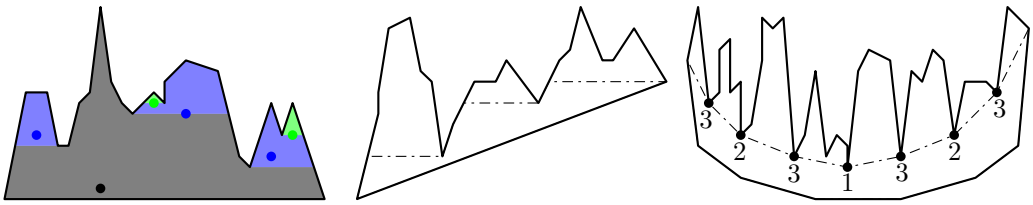


Figure 7 x-monotone polygons over a single horizontal edge, a sloped edge and a convex chain.

Now let's look at a monotone polygon over a single non-horizontal edge, without loss of generality ascending to the right. We show in the middle picture of Figure 7 that P_n can be partitioned into a set of independent monotone polygons over a horizontal edge and a tilted monotone polygon over a horizontal edge. Hence by Lemma 2 for any such polygon we have $\chi(P_n) \leq 2g(n) \leq 2 \log n$.

Monotone polygons P_n over a convex subchain are also easily covered with $O(\log n)$ colors. The shortest path in P_n from the leftmost vertex to the rightmost vertex cuts off independent monotone polygons over a single edge. The remaining subpolygon is bounded by a concave chain on top and the convex chain at the bottom. We can cover such a polygon using $\log n$ colors, by the following recursive process: place a guard of color $i = 1$ at the middle vertex of the concave chain; increment the color to $i = 2$, place guards of color 2 at the middle vertex of the two subchains, and so on. Clearly this requires $\log n$ colors, so it remains to show that the polygon is covered and the coloring is conflict-free. Let p be a point in the remaining subpolygon and let $l(p)$ be the list of all guard colors p can see. Between any two guards on the concave subchain that have the same color there must lie a guard of lower color between them. Hence the minimal color in $l(p)$ is a unique color among all guards that contain p in their visibility region. Therefore by Lemma 2, for any monotone polygon P_n over a convex chain we have $\chi(P_n) \leq 2g(n) + \log n \leq 3 \log n$.

Monotone polygons over a concave subchain

For monotone polygons P_n over a concave chain, we cut off independent monotone subpolygons over a horizontal edge as we did before in the case of a non-horizontal base edge. This

results in two additional independent subpolygons whose boundary consists of a lower subchain which is concave and strictly increasing (respectively strictly decreasing) and an upper subchain which is monotonically increasing (respectively monotonically decreasing). In both of these subpolygons we place colored guards on the concave subchains as we did in the case of monotone subpolygons over a convex subchain. We show the partition and the guard placement and coloring in Figure 8. Let P be the subpolygon over the strictly increasing concave subchain. If a point p in P is guarded by a guard on the concave subchain, it has a guard of unique color among all other guards on the concave subchain that see p. However, there may be regions in P not guarded by the guards on the concave subchain. For these regions we have the following technical lemma, whose proof is omitted due to lack of space.

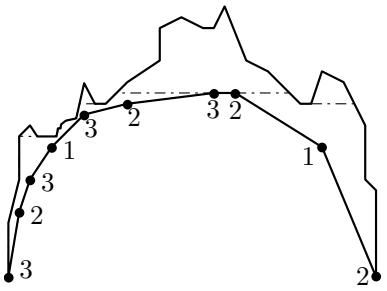


Figure 8 Guard placement for monotone polygon over a concave subchain.

▶ **Lemma 9.** *If a point $p \in P$ is not visible from any of the guards on the concave subchain, then p lies in a not yet guarded simply connected region, which has the shape of a monotone subpolygon over a convex chain. Furthermore all such regions are independent.*

Thus, we have a partition into monotone polygons over a single horizontal edge (where we need at most $\log n$ colors), monotone polygons over a convex chain (at most $3 \log n$ colors) and the two independent subpolygons guarded by the guards on the concave chain (at most $\log n$ colors). By Lemma 2 we have that for any monotone polygon P_n over a concave chain, $\chi(P_n) \leq 5 \log n$. In view of Lemma 8, we now have the main result of this section.

▶ **Theorem 10.** *The conflict-free chromatic guard number for monotone polygons on n vertices is $k(n) = O(\log n)$.*

5 Arbitrary Simple Polygons

Our proof structure for orthogonal polygons has the following form. We first partitioned the polygon into four different types of subpolygons and showed that the process terminates after a finite number of steps (Lemma 3). We then derived a necessary condition for two subpolygons of the same type not to be independent (Lemma 4). We then found a conflict-free covering using three colors for the basic building blocks, the staircase polygons (Lemma 5). We used this to get an upper bound of $4 \log n$ for polygons corresponding to red-black-alternating subtrees (Lemma 6). Finally we put all subtrees together to achieve an $O(\log n)$ upper bound on the chromatic guard number $k(n)$ for orthogonal polygons.

Our proof for non-orthogonal simple polygons follows the same outline, with appropriate differences spelled out. Specifically, given a n-vertex polygon P_n, we construct a partition $\{A_U, A_D, A_L, A_R\}$, where A_U, A_D, A_L, A_R, respectively, is the collection of up-polygons (depicted in red), down-polygons (in green), left-polygons (in black) and right-polygons (in blue). We rotate P_n in such a way that we can start with a horizontal line segment which gives rise to a first subpolygon of type U. Since the polygon's edges are no longer axis parallel, the partitioning process can be trapped between to edges e and f that ascend to the same direction. This gives rise to a long and possibly infinite alternating path, see the left picture in Figure 9.

In order to deal with this difficulty, we do the following. When an edge e of P gets touched during the partitioning process for the *second time* by a subpolygon of the same

type, *without any vertex being touched in the meantime*, we extend this subpolygon until it touches a vertex of e or f, see the right picture in Figure 9. This ensures that in at least

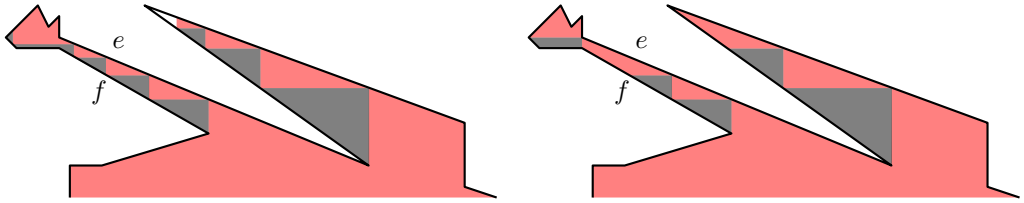


Figure 9 Replacing a red-black-alternating path with an augmented U subpolygon.

every third step of the partitioning process a vertex gets touched. Now any vertex v can be touched at most five times, since after the first time v gets touched, in each following step at least an additional 90° of v's interior angle are covered by a subpolygon of the partition. Along the lines of Lemma 3, this modification allows us to prove the following result, whose proof is omitted from this extended abstract due to lack of space.

▶ **Lemma 11.** *The revised partitioning process gives a complete partition after a finite number of (at most* $15n$*) steps.*

This replacement of alternating paths with a single polygon slightly changes the definition of the subpolygon types in the partitioning, but its does not change the relations between subpolygons of the same type when it comes to visibility—we simply replaced an alternating path with a *shorter* alternating path. This means that the schematic tree of the revised partitioning process has the same properties as the original partitioning process in orthogonal polygons, in particular we get as a corollary from Lemma 4:

▶ **Lemma 12.** *Let P be a polygon with the given revised partition and the schematic tree T. Let P_i and P_j be two arbitrary subpolygons of type U. Then, either (i) P_i and P_j are independent, or (ii) there exists a red-black-alternating (or a red-blue-alternating) directed path in T between P_i and P_j.*

This allows us to invoke the same coloring strategy as used in orthogonal polygons. We first focus on polygon regions corresponding to red-black-alternating trees. A polygon P_n corresponding to a red-black-alternating tree consists of type U and type L subpolygons; it "grows to the left". In place of Lemma 5, which states a constant conflict-free chromatic guard number for staircase polygons, we have Theorem 10, which gives an $O(\log n)$ for monotone polygons. We cover a part of P_n with a polygon that is both x- and y-monotone: Starting from the lowest and rightmost vertex of P_n, at every decision line we follow the upper part with the most vertices. We need $O(\log n)$ colors to do this plus an additional color to cover the convex fans to its left as before. We are left with independent subtrees, all of size $\leq \lfloor n/2 \rfloor$. We recursive each of them and cover all type U subpolygons of P_n in at most $\log n$ rounds. This leads to the following result.

▶ **Lemma 13.** *Suppose P_n is a simple polygon with a partition that has a red-black-alternating schematic tree. Then a conflict-free coloring of all the red regions of P_n needs at most $O(\log^2 n)$ colors. The same bound also holds for a red-blue-alternating schematic tree.*

The composition of red-black-alternating trees and red-blue-alternating trees that we described earlier depended only on the condition of Lemma 4, which we preserved in the revised partition of arbitrary polygons, see Lemma 12. Considering this, we can put subtrees

together as we did in the case of orthogonal polygons. Thus we finally get an upper bound for simple polygons.

▶ **Theorem 14.** *The conflict-free chromatic guard number for simple non-orthogonal polygons on n vertices is $k(n) = O(\log^2 n)$.*

6 Conclusions

The art gallery problems provide a conceptually clean and mathematically elegant framework to study many applied questions related to surveilling, monitoring and covering of a physical environment. In this paper, we studied a *chromatic* variant of the art gallery, where the primary concern is to minimize the number of distinct *colors* assigned to guards. Our two main results are that (i) every n-vertex simple polygon has a conflict-free chromatic art gallery coverage with $O(\log^2 n)$ colors, and (ii) if the polygon is orthogonal, then the number of colors is only $O(\log n)$. A stronger form of coloring, which requires all guards visible to a point to be distinct in colors, needs $\Omega(n)$ colors for simple polygons and $\Omega(\sqrt{n})$ for orthogonal polygons [3], showing that the weaker conflict-free condition gives a significant improvement in the number of colors.

Our work suggests several directions for future research. Perhaps the most natural question is to investigate the lower bounds on the number of colors needed. Currently, we have none. What is the tight bound for the simple non-orthogonal polygons? Finally, the line-of-sight visibility model is a crude model for wireless communication. Recently, Fabila-Monroy et al. [5] have investigated the art gallery problems that allows the signal to penetrate k walls. One could consider our chromatic art gallery in a similar setting.

Acknowledgment We thank Luca Foschini for some insightful discussions during this research.

References

1 A. Bärtschi. Coloring Variations of the Art Gallery Problem. Master's thesis, Department of Mathematics, ETH Zürich, August 2011.
2 V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39 – 41, 1975.
3 L. H. Erickson and S. M. LaValle. An art gallery approach to ensuring that landmarks are distinguishable. In *Proc.of Robotics: Science and Systems*, June 2011.
4 L. H. Erickson and S. M. LaValle. A chromatic art gallery problem. Technical report, Departement of Computer Science, University of Illinois at Urbana-Champaign, 2011.
5 R. Fabila-Monroy, A. Ruiz-Vargas, and J. Urrutia. On Modem Illumination Problems. In *Proc. XIII Encuentros de Geometria Computacional*, Zaragoza, Spain, June 2009.
6 J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987.
7 J. Pach and G. Tardos. Coloring axis-parallel rectangles. *Journal of Combinatorial Theory, Series A*, 117(6):776–782, 2010.
8 T.C. Shermer. Recent results in art galleries. *Proc. of the IEEE*, 1992.
9 S. Smorodinsky. *Combinatorial Problems in Computational Geometry*. PhD thesis, School of Computer Science, Tel-Aviv University.
10 S. Smorodinsky. On the chromatic number of some geometric hypergraphs. In *Proc. of 17th ACM-SIAM Symposium on Discrete Algorithms*, pages 316–323, 2006.
11 S. Smorodinsky. Conflict-Free Coloring and its Applications, 2010. `http://arxiv.org/abs/1005.3616v2`.

Constant compression and random weights*

Wolfgang Merkle[1] and Jason Teutsch[2]

1,2 Ruprecht-Karls-Universität Heidelberg
Heidelberg, Germany
{merkle|teutsch}@math.uni-heidelberg.de

Abstract

Omega numbers, as considered in algorithmic randomness, are by definition real numbers that are equal to the halting probability of a universal prefix-free Turing machine. Omega numbers are obviously left-r.e., i.e., are effectively approximable from below. Furthermore, among all left-r.e. real numbers in the appropriate range between 0 and 1, the Omega numbers admit well-known characterizations as the ones that are Martin-Löf random, as well as the ones such that any of their effective approximation from below is slower than any other effective approximation from below to any other real, up to a constant factor. In what follows, we obtain a further characterization of Omega numbers in terms of Theta numbers.

Tadaki considered for a given prefix-free Turing machine and some natural number a the set of all strings that are compressed by this machine by at least a bits relative to their length, and he introduced Theta numbers as the weight of sets of this form. He showed that in the case of a universal prefix-free Turing machine any Theta number is an Omega number and he asked whether this implication can be reversed. We answer his question in the affirmative and thus obtain a new characterization of Omega numbers.

In addition to the one-sided case of the set of all strings compressible by at least a certain number a of bits, we consider sets that comprise all strings that are compressible by at least a but no more than b bits, and we call the weight of such a set a two-sided Theta number. We demonstrate that in the case of a universal prefix-free Turing machine, for given a and all sufficiently large b the corresponding two-sided Theta number is again an Omega number. Conversely, any Omega number can be realized as two-sided Theta number for any pair of natural numbers a and $b > a$.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases computational complexity, Kolmogorov complexity, algorithmic randomness, Omega number

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.172

1 Universal prefix-free machines and random reals

An Omega number is the weight of the domain of a universal prefix-free machine U, i.e., a real number of the form $\Omega_U = \sum_{\sigma \in \operatorname{dom} U} 2^{-|\sigma|}$. Chaitin [3] introduced Omega numbers and demonstrated, after work of Zvonkin and Levin [10], that Omega numbers admit recursive approximations from below yet feature completely random binary expansions. That is, Omega numbers are left-r.e. and Martin-Löf random. Remarkably, Omega numbers are the only such numbers and therefore characterize the set of reals with these two properties. Calude, Hertling, Khoussainov, Wang [1] and Kučera, Slaman [5] proved this equivalence known as the Kučera-Slaman Theorem [4, p. 410].

* Research supported by Deutsche Forschungsgemeinschaft under grant ME 1806/3-1.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 172–181

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

Tadaki introduced Theta numbers Θ^a_M as the weight of the set of strings that can be compressed by a constant number a of bits relative to their length with respect to the coding given by some prefix-free Turing machine M, i.e.,

$$\Gamma^a_M = \{\sigma \in \{0,1\}^* : (\exists \tau)\, M(\tau) = \sigma \text{ and } |\tau| \leq |\sigma| - a\}, \qquad \Theta^a_M = \sum_{\sigma \in \Gamma^a_M} 2^{-|\sigma|}.$$

Tadaki showed that in the case of a prefix-free universal Turing machine any Theta number is an Omega number and he asked whether this implication can be reversed. We answer his question in the affirmative and obtain this way a new characterization of Omega numbers.

In addition to the one-sided case of the set Γ^a_M of all strings compressible by at least a of bits, we consider the two-sided case of the set $\Gamma^{a\setminus b}_M$ of all strings that are compressible by at least a but no more than b bits. We demonstrate that such sets cannot contain an r.e. set, hence are not r.e., but somewhat surprisingly, in the case of a universal prefix-free machine, for given a and for all sufficiently large b the corresponding two-sided Theta number $\Theta^{a\setminus b}_M$ is left-r.e. and, in fact, is again an Omega number. Conversely, any Omega number can be realized as two-sided Theta number for any pair of natural numbers a and $b > a$.

Note that due to space considerations in the sequel several results are stated without proof.

Notation A STRING is a finite binary sequence, the length of a string σ is denoted by $|\sigma|$, where $|\cdot|$ will also denote cardinality for sets. A set of strings is PREFIX-FREE if no string in the set is a proper prefix of another string in the set. We let $\operatorname{dom} M = \{\sigma : M(\sigma)\downarrow\}$ be the domain of a Turing machine M, where $M(\sigma)\downarrow$ and $M(\sigma)\uparrow$ denote convergence and divergence of the computation of M on input σ. A prefix-free Turing machine or, for short, a PREFIX-FREE MACHINE is a Turing machine that has prefix-free domain. The PREFIX-FREE KOLMOGOROV COMPLEXITY of a string σ with respect to a prefix-free machine M, denoted K_M, is the length of the shortest input to M which results in output σ. A prefix-free machine U is UNIVERSAL if for any other prefix-free machine M, there exists a constant c such that for all strings σ, $\mathrm{K}_U(\sigma) \leq \mathrm{K}_M(\sigma) + c$. Universal prefix-free machines exist [4, 6]. We fix some reference universal prefix-free machine U and write K in place of K_U. Furthermore, let $\mathrm{K}(\sigma|\rho) = \min\{|\tau| : U(1^{|\rho|}0\rho\tau) = \sigma\}$ denote the PREFIX-FREE COMPLEXITY OF σ GIVEN ρ.

We will identify strings and natural numbers via the order morphisms between the length-lexicographical ordering on strings and the usual order on the natural numbers, and accordingly the function K, in addition to strings, may take natural numbers as arguments or even integers, where the latter are viewed as a coded pair of a natural number and the sign. For a natural number n, we let n^* denote a code for n of minimum length, i.e., $U(n^*) = n$ and $|n^*| = \mathrm{K}(n)$, where among all codes of minimum length the code n^* is the one with the least running time on U, breaking ties by choosing the least string in lexicographical order. For $n = 0, 1, \ldots$, we let $\bar{n}$ denote an encoding of the natural number n with respect to U, i.e, $U(\bar{n}) = n$, that has length at most $2\log n + c$ for some constant c [4], where log denotes logarithm to base 2. We choose the mapping $n \mapsto \bar{n}$ to be recursive, while this would not be possible for the mapping $n \mapsto n^*$.

Unless explicitly specified otherwise, the term SEQUENCE refers to an infinite binary sequence. A sequence $x_1x_2\ldots$ can be viewed as the real that has binary expansion $0.x_1x_2\ldots$, and the notation sequence and real will be used interchangeably. A real number $\alpha \in [0,1]$ is called LEFT-R.E. if it is the limit of an effectively given sequence of nonnegative dyadic rationals, i.e., nonnegative rationals with denominators that are a power of 2. For a real α

equal to $0.x_1x_2\ldots$, the string $\alpha \upharpoonright n$ consists of the first n bits $x_1x_2\ldots x_n$ of α after the decimal point. A real α is MARTIN-LÖF RANDOM if there exists a constant c such that $\mathrm{K}(\alpha \upharpoonright n) \geq n - c$ for all n. This definition coincides with our intuition that random objects do not compress too much. A real that is left-r.e. and Martin-Löf random is called an OMEGA NUMBER. For further background on notions discussed in this section, see the monograph by Downey and Hirschfeldt [4], which contains also a detailed account of the Kraft-Chaitin theorem to be used in the sequel.

2 Sets of compressible strings

Kolmogorov complexity comes in several flavors [4]. Besides the prefix-free Kolmogorov complexity K introduced in Section 1, one can consider the PLAIN version defined similarly but without any requirements on machines being prefix-free. The plain Kolmogorov complexity of a string of length n never exceeds n plus an additive constant, and a straightforward combinatorial argument shows that for some positive constant d and all natural numbers a and n, at most a fraction of 2^{-a+d} of all strings of length n have plain Kolmogorov complexity of at most $n - a$ [4, p. 112]. For prefix-free Kolmogorov complexity, the situation is similar but somewhat more involved because the upper bound of n has to be replaced by $n + \mathrm{K}(n)$. Prefix-free Kolmogorov complexity for strings of length n may achieve but never exceeds $n + \mathrm{K}(n)$, up to an additive constant [4, p. 128]. Furthermore, Chaitin's celebrated Counting Theorem asserts that the number of strings describable by codes shorter than this upper bound minus a constant has a simple upper bound reminiscent of the one for plain complexity.

▶ **Counting Theorem** (Chaitin [2, 4, 6]). *For some positive constant d and all natural numbers a and n, it holds that*

$$|\{\sigma \in \{0,1\}^n \colon \mathrm{K}(\sigma) \leq n + \mathrm{K}(n) - a\}| \leq 2^{n-a+d} \,,$$

i.e., at most a fraction of 2^{-a+d} of all strings of length n have prefix-free Kolmogorov complexity of no more than $n + \mathrm{K}(n) - a$.

When working with plain Kolmogorov complexity, it is suggestive to call an n-bit string a-compressible in case the plain Kolmogorov complexity of the string is at most $n - a$ and to call a string compressible in case it is 1-compressible. Following the literature conventions, we extend this notation to the prefix-free setting. Note that indicating compression relative to n and not relative to the upper bound $n + \mathrm{K}(n)$ avoids having to count bits of compression relative to the nonrecursive latter bound. We will, by slight abuse of notation, permit our notation to carry over to negative values of a because some of our results extend to this case.

▶ **Definition 1.** Let a be any integer. A string σ is a-COMPRESSIBLE WITH RESPECT TO A PREFIX-FREE MACHINE M if $\mathrm{K}_M(\sigma) \leq |\sigma| - a$. Furthermore, a string σ is a-COMPRESSIBLE if $\mathrm{K}(\sigma) < |\sigma| - a$.

▶ **Definition 2.** Let M be a prefix-free machine and let a and b be integers. The SET OF a-COMPRESSIBLE STRINGS WITH RESPECT TO M, denoted Γ^a_M, is

$$\Gamma^a_M = \{\sigma \in \{0,1\}^* : (\exists \tau)\, M(\tau) = \sigma \text{ and } |\tau| \leq |\sigma| - a\},$$

and the SET OF $[a,b)$-COMPRESSIBLE STRINGS WITH RESPECT TO M is

$$\Gamma^{a\setminus b}_M = \Gamma^a_M - \Gamma^b_M \,.$$

We will refer to sets of the form Γ^a_M and $\Gamma^{a\setminus b}_M$ as ONE-SIDED and TWO-SIDED GAMMA SETS, respectively, and we will call such Gamma sets UNIVERSAL in case M is a universal prefix-free machine.

By the Counting Theorem, there exists a constant d such that for all integers a and string lengths n,

$$|\Gamma^a_U \cap \{0,1\}^n| \leq 2^{n-\mathrm{K}(n)-a+d}\,. \tag{1}$$

In particular, this shows that for any integer a the fraction of strings of length n that are a-compressible goes to 0 when n goes to infinity.

Miller and Yu [7] refined Chaitin's Counting Theorem [4, Section 3.7]. Using their result, we can improve (1) to lower and upper bounds that match up to a constant factor. We state Miller and Yu's result in a slightly altered form where we replace one occurrence of $\mathrm{K}(\sigma)$ by $\mathrm{K}_U(\sigma)$ for an arbitrary universal prefix-free machine U. One can resolve the differences introduced in our alternate version via appropriately chosen values for the constant d; details are left to the reader.

▶ **Improved Counting Theorem** (Miller and Yu [7]). *Let U be a universal prefix-free machine. There is a constant d such that for all natural numbers c and n it holds that*

$$2^{n-c-\mathrm{K}(c|n^*)-d} \leq |\{\sigma \in \{0,1\}^n : \mathrm{K}_U(\sigma) \leq n + \mathrm{K}(n) - c\}| \leq 2^{n-c-\mathrm{K}(c|n^*)+d}\,.$$

Note that the bounds given by the Improved Counting Theorem are false in general for negative values of c. By the Improved Counting Theorem we obtain in Corollaries 3 and 4 bounds for the number of strings of length n in sets of the form $\Gamma^{a\setminus b}_U$ and $\Gamma^{a\setminus b}_U$ for a universal prefix-free machine U. Proposition 5 then shows that the assertion of Corollary 4 cannot be strengthened to hold for all b instead of just all sufficiently large b.

▶ **Corollary 3.** *Let U be any universal prefix-free machine. There is a constant d such that for all natural numbers a and all n, as well as for all integers a and for all sufficiently large natural numbers n it holds that*

$$2^{n-\mathrm{K}(n)-a-\mathrm{K}(a|n^*)-d} \leq |\Gamma^a_U \cap \{0,1\}^n| \leq 2^{n-\mathrm{K}(n)-a-\mathrm{K}(a|n^*)+d}\,. \tag{2}$$

▶ Remark. Tadaki states the special case of Corollary 3 where a is equal to 1 and attributes this result to Solovay [9, Theorem 5]. For this special case, as with any constant value of a, the additive terms a and $\mathrm{K}(a|n^*)$ in the exponents of the bounding terms in (2) can be subsumed into the constant d.

▶ **Corollary 4.** *Let U be any universal prefix-free machine and let a be any integer. Then for any real $\varepsilon > 0$, for all sufficiently large integers b, and for all n it holds that*

$$(1-\varepsilon)\,|\Gamma^a_U \cap \{0,1\}^n| \leq \left|\Gamma^{a\setminus b}_U \cap \{0,1\}^n\right| \leq |\Gamma^a_U \cap \{0,1\}^n|\,. \tag{3}$$

▶ **Proposition 5.** *For every pair of integers a and b there is a universal prefix-free machine U such that $\Gamma^{a\setminus b}_U$ is empty.*

3 Compressible strings and enumerability

We note that by definition every one-sided Gamma set is r.e., and every two-sided Gamma set is the difference of two r.e. sets, or D.R.E., for short (see the monographs cited in the

references [4, 6, 8] for background on r.e. and d.r.e. sets). Furthermore, in general sets of the form Γ^a_M and $\Gamma^{a\setminus b}_M$ can be rather simple and may for example be empty or may be infinite and recursive, where the latter can be achieved by choosing M to be a prefix-free machine where $M(0^k) = 0^{k+1}$ while M is undefined, otherwise. In contrast to this, complements of one-sided universal Gamma sets cannot even be r.e because any infinite r.e. set must contain highly compressible strings. For two-sided Gamma sets $\Gamma^{a\setminus b}_U$ a similar assertion holds for sufficiently large b according to Proposition 6. We conclude this section by Lemma 7 which provides the technical machinery for Theorems 10 and 11.

▶ **Proposition 6.** *Let U be a universal prefix-free machine, and let a be any integer. For any integer b, the set $\Gamma^{a\setminus b}_U$ does not contain an infinite r.e. set. For almost all integers $b > a$, the complement of the set $\Gamma^{a\setminus b}_U$ is not r.e.*

▶ **Lemma 7.** *Let U be a universal prefix-free machine and let a and b be any integers where $a < b$. Suppose that for each integer t an enumeration without repetitions of the set Γ^t_U is given uniformly effectively in t and let $\sigma_0, \sigma_1, \sigma_2, \ldots$ and $\tau_0, \tau_1, \tau_2, \ldots$ be the corresponding enumerations of Γ^a_U and Γ^b_U, respectively. Furthermore, let d be any natural number and let r be any recursive function. Then for all sufficiently large b there is a strictly increasing recursive function g such that for all i,*

(I) $\left|\sigma_{g(i)}\right| = |\tau_i| - d$,

(II) $g(0) > r(0)$ *and* $g(i+1) > r(g(i))$,

(III) $\sigma_{g(i)} \neq \tau_j$ *for* $j = 0, \ldots, r(i)$.

Proof. Each of the b-compressible strings τ_j occurs exactly once in the sequence $\sigma_0, \sigma_1, \ldots$, thus there is a computable function h such that for all i, each of the strings $\tau_0, \ldots, \tau_i$ occurs among the strings $\sigma_0, \ldots, \sigma_{h(i)}$, hence does not occur among $\sigma_{h(i)+1}, \sigma_{h(i)+2}, \ldots$. For further use note that $h(i) \geq i$.

We define inductively functions γ and g, which *a priori* are not necessarily total. For a start, we set m_0 to $h(r(0)) = \max\{r(0), h(r(0))\}$, let $\gamma(0)$ be the least string of length $|\tau_0| - d$ that differs from σ_0 through σ_{m_0}, and let $g(0)$ be the least (in fact possibly undefined but if defined unique) index j such that $\gamma(0) = \sigma_j$. Assuming that γ and g have already been defined for all arguments up to i, let

$$\begin{aligned} m_{i+1} &= \max\{g(i), r(g(i)), h(r(i))\}, \\ \gamma(i+1) &= \min\left\{\eta \in \{0,1\}^{|\tau_{i+1}|-d} \colon \eta \neq \sigma_j \text{ for } j \in \{0, \ldots, m_{i+1}\}\right\}, \\ g(i+1) &= \min\{j \colon \sigma_j = \gamma(i+1)\}, \end{aligned}$$

that is, $\gamma(i+1)$ is the lexicographically least string of length $|\tau_{i+1}| - d$ that differs from all the strings $\sigma_0, \ldots, \sigma_{m_{i+1}}$, while $g(i+1)$ is the index of $\gamma(i+1)$ in the enumeration $\sigma_0, \sigma_1, \ldots$.

Now consider any i such that $\gamma(i)$ and $g(i)$ are both defined. Then assertion (I) holds true by choice of $\gamma(i)$ and because $\gamma(i)$ and $\sigma_{g(i)}$ are the same. Furthermore, assertions (II) and (III) hold true because of $g(i) > m_i$ and because by choice of h and m_i, $\sigma_{g(i)}$ differs from τ_0 through $\tau_{r(i)}$. Since the functions γ and g are partial recursive, in order to prove the lemma, it remains to show that g is total for all sufficiently large b.

By the Counting Theorem, for some n_0 and all $n \geq n_0$ there exists a string of length $n-d$ that is not a-compressible. In case $b > n_0$, the b-compressible strings $\tau_0, \tau_1, \ldots$ must all have length at least n_0, hence when trying to define $\gamma(i+1)$ there will always be a string of length $|\tau_{i+1}| - d$ that differs from the a-compressible strings σ_0 through $\sigma_{m_{i+1}}$. So in case $b > n_0$, the only way g might avoid being total is that there is a least index i such that

the functions g and γ are defined on all values up to i, the string $\gamma(i+1)$ is defined, too, but the value $g(i+1)$ is undefined. That is, the string $\gamma(i+1)$ is defined but does not occur in the enumeration $\sigma_0, \sigma_1, \ldots$ of all a-compressible strings, which we show is impossible.

Consider a prefix-free machine M that assumes its input to be of the form $\bar{a}\bar{b}\rho$ where a and b are integers and ρ is a prefix-free code for some b-compressible string, i.e., $U(\rho) = \tau_i$ for some index i. In case M is able to verify this assumption, M simulates the inductive definition of γ and g in order to compute $\gamma(i+1)$, and outputs $\eta = \gamma(i+1)$. But then there exists a c such that for all large enough b and for an optimal code ρ for τ_i,

$$\mathrm{K}_U(\eta) \leq \left|\bar{a}\bar{b}\rho\right| + c \leq \left|\bar{a}\bar{b}\right| + |\tau_i| - b + c = |\tau_i| - d - a - (b-a) + \left|\bar{a}\bar{b}\right| + c + d \leq |\eta| - a,$$

where the inequalities follow, first, by universality of U, second, by choice of ρ as a code of length at most $|\tau_i| - b$, third, by rearranging terms, and, last, because η has length $|\tau_i| - d$ and because for any b that is large enough the difference $b - a$ will be larger than $\left|\bar{a}\bar{b}\right|$ plus the constant $c + d$. Hence for sufficiently large b, for all i the string $\gamma(i)$ is a-compressible, hence $g(i)$ is defined. ◀

4 Left-r.e. approximations for Theta numbers

In the following definition, we review and slightly extend Tadaki's [9] concept of Theta number, which is central for this exposition.

▶ **Definition 8.** The WEIGHT of a (not necessarily finite) set A of strings is the value of the sum $\sum_{\sigma\in A} 2^{-|\sigma|}$, and the WEIGHT of a singleton string σ is $2^{-|\sigma|}$. For a prefix-free machine M and integers a and b let

$$\Theta^a_M = \sum_{\sigma\in\Gamma^a_M} 2^{-|\sigma|} \quad \text{and} \quad \Theta^{a\setminus b}_M = \sum_{\sigma\in\Gamma^{a\setminus b}_M} 2^{-|\sigma|}$$

be the weights of the set Γ^a_M of a-compressible strings and of the set $\Gamma^{a\setminus b}_M$ of $[a,b)$-compressible strings with respect to M.

We will refer to reals of the form Θ^a_M and $\Theta^{a\setminus b}_M$ as ONE-SIDED and TWO-SIDED THETA NUMBERS, respectively. A Theta number is UNIVERSAL if its underlying prefix-free machine is universal. Note that for any prefix-free machine M and any integers a and b, in case $a \leq b$, we have Γ^b_M is a subset of Γ^a_M and therefore

$$\Theta^{a\setminus b}_M = \Theta^a_M - \Theta^b_M\,,$$

whereas $\Theta^{a\setminus b}_M = 0$, otherwise. Furthermore, for any prefix-free machine M and any integers a and b, the Theta numbers Θ^a_M and $\Theta^{a\setminus b}_M$ are both finite since both can be at most as large as 2^{-a} times the weight of the domain of the prefix-free machine M, where the latter weight is at most 1 by the Kraft inequality, i.e.,

$$\Theta^{a\setminus b}_M \leq \Theta^a_M = \sum_{\tau\in\Gamma^a_M} 2^{-|\tau|} \leq \sum_{\sigma\in\mathrm{dom}\,M} 2^{-(|\sigma|+a)} \leq 2^{-a}\,. \tag{4}$$

As observed by Tadaki [9], the real Θ^1_U and indeed all one-sided Theta numbers, or reals of the form Θ^a_M for integers a, are the weight of some r.e. set, which is equivalent to being left-r.e. Proposition 6, which says that $\Gamma^{a\setminus b}_M$ need not be r.e., now comes back to haunt us because in contrast to the one-sided case, a two-sided Theta number may fail to be left-r.e.

▶ **Proposition 9.** *Let a be any integer. There exists a prefix-free machine M such that for all $b > a$ the real $\Theta_M^{a\setminus b}$ is not left-r.e.*

The following theorem asserts that two-sided Theta numbers $\Theta_U^{a\setminus b}$ are indeed left-r.e for all sufficiently large b in the case of a universal prefix-free machine U. This result comes as a slight surprise since for all sufficiently large b the set $\Gamma_U^{a\setminus b}$ is not r.e. according to Proposition 6.

▶ **Theorem 10.** *Let U be a universal prefix-free machine, and let a be any integer. For all sufficiently large integers b, the real $\Theta_U^{a\setminus b}$ is left-r.e.*

Proof. Apply Lemma 7 to U and a where d is equal to 0 and r is the identity function. Fix any b that is so large that there are enumerations $\sigma_0, \sigma_1, \sigma_2, \dots$ and $\tau_0, \tau_1, \tau_2, \dots$ of Γ_U^a and Γ_U^b, respectively, and a recursive function g as in Lemma 7. Recall that the function g is strictly increasing, hence is one-to-one and its range R is recursive. Then $\Theta_U^{a\setminus b}$ is left-r.e. because we have

$$\begin{aligned}\Theta_U^{a\setminus b} &= \sum_{\sigma\in\Gamma_M^{a\setminus b}} 2^{-|\sigma|} = \sum_{\sigma\in\Gamma_M^a} 2^{-|\sigma|} - \sum_{\tau\in\Gamma_M^b} 2^{-|\tau|}\\ &= \sum_{k\in\mathbb{N}\setminus R} 2^{-|\sigma_k|} + \sum_{k\in\mathbb{N}\cap R} 2^{-|\sigma_k|} - \sum_{k\in\mathbb{N}} 2^{-|\tau_k|}\\ &= \sum_{k\in\mathbb{N}\setminus R} 2^{-|\sigma_k|} + \sum_{k\in\mathbb{N}} \underbrace{2^{-|\sigma_{g(k)}|} - 2^{-|\tau_k|}}_{=0}.\end{aligned}$$

◀

5 Theta numbers and Martin-Löf randomness

Tadaki [9] demonstrated that every one-sided universal Theta number is Martin-Löf random. Using Theorem 10, we extend Tadaki's result to show that two-sided universal Theta numbers are Martin-Löf random. As just mentioned, the first statement in the following theorem is due to Tadaki [9].

▶ **Theorem 11.** *Let U be a universal prefix-free machine and let a be a natural number.*

(I) *The real Θ_U^a is Martin-Löf random.*

(II) *For all sufficiently large natural numbers b, the real $\Theta_U^{a\setminus b}$ is Martin-Löf random.*

Proof of (II). Fix any natural number $b > a$. Assuming that $\Theta_U^{a\setminus b}$ is not Martin-Löf random, we will obtain a contradiction if b is sufficiently large. In order to apply Lemma 7, take $d = 1$ and let r be equal to the identity function. Furthermore, let $\sigma_0, \sigma_1, \sigma_2, \dots$ and $\tau_0, \tau_1, \tau_2, \dots$ be enumerations of Γ_U^a and of Γ_U^b, respectively, as in the assumption of the lemma. Then for sufficiently large b there is a strictly increasing function g as in the lemma, i.e., for all i, the string $\sigma_{g(i)}$ is one bit shorter than the string τ_i and differs from $\tau_0, \tau_1, \dots, \tau_i$. Next let for any natural number s,

$$I_s = \{i \le s \colon \sigma_i \notin \{\tau_0, \dots \tau_s\}\} \quad \text{and} \quad \Theta_{U,s}^{a\setminus b} = \sum_{i\in I_s} 2^{-|\sigma_i|}\,.$$

Observe that the sequence $\{\Theta_{U,s}^{a\setminus b}\}$ converges to $\Theta_U^{a\setminus b}$, but not necessarily monotonically so. Similarly to the one-sided case, let M be a prefix-free machine that, on input η, first tries to compute the string $U(\eta)$ and its length n. If successful, M next searches for the least s such

that the length n initial segment of the binary expansion of $\Theta_{U,s}^{a\setminus b}$ is equal to $U(\eta)$. If such a number s is found, M outputs the least string of length $n-2$ that differs from $\sigma_0, \ldots, \sigma_s$, where such an output string exists for all sufficiently large n by the Counting Theorem.

By letting d_0 be equal to the coding constant for M with respect to U, we can fix a sufficiently large length n such that the following holds. The initial segment of $\Theta_U^{a\setminus b}$ of length n is equal to $U(\eta)$ for some code η of length at most $n-a-d_0-2$, and the string $M(\eta)$ exists, has length $n-2$, and satisfies $\mathrm{K}(M(\eta)) \leq |\eta| + d_0 \leq n-a-2$. It follows that $M(\eta)$ is a-compressible, hence is equal to σ_t for some index $t > s$.

For the length n indicated in the previous paragraph, consider the corresponding values of s, η and t and the corresponding set I_s, as well as the set $I_s^+ = I_s \cup \{t\}$. By choice of η, the set I_s^+ contains only indices of a-compressible strings and the sum of the weights of these strings is strictly larger than $\Theta_U^{a\setminus b}$. More precisely, since $\Theta_{U,s}^{a\setminus b}$ differs from $\Theta_U^{a\setminus b}$ by at most 2^{-n}, we have

$$\sum_{i\in I_s^+} 2^{-|\sigma_i|} = \left(\sum_{i\in I_s} 2^{-|\sigma_i|}\right) + 2^{-|\sigma_t|} = \Theta_{U,s}^{a\setminus b} + 4\cdot 2^{-n} \geq \Theta_U^{a\setminus b} + 3\cdot 2^{-n}\,. \tag{5}$$

Having the weight of strings indexed by I_s^+ to be greater than $\Theta_U^{a\setminus b}$ is not a contradiction because some of these strings may in fact be b-compressible and hence do not contribute to $\Theta_U^{a\setminus b}$. However, whenever a string ρ is b-compressible, i.e., is equal to some string τ_j, then $\sigma_{g(j)}$ is another a-compressible string with strictly greater weight than ρ. The string $\sigma_{g(j)}$ may be b-compressible in turn, in which case there is another a-compressible string of weight strictly larger than $\sigma_{g(j)}$. Iterating this process, we eventually reach a TERMINAL a-compressible string that is not b-compressible and contributes its weight to $\Theta_U^{a\setminus b}$. In the remainder of the proof, we argue that the terminal strings that are reached by such cascades starting from strings with indices in I_s^+ have a total weight that is strictly larger than $\Theta_U^{a\setminus b}$, which is then indeed a contradiction.

Formally, define a partial function h such that σ_i is equal to $\tau_{h(i)}$ in case σ_i is indeed b-compressible, and h is undefined otherwise. Let $f = g \circ h$. Then for all i such that $h(i)$ is defined, we have

$$\sigma_{f(i)} = \sigma_{g(h(i))}, \quad \text{hence} \quad |\sigma_{f(i)}| = |\sigma_i| - 1 \quad \text{by choice of } h \text{ and } g.$$

▶ Claim 1. The function f is one-to-one in the sense that if $f(i)$ and $f(j)$ are both defined and are the same, then i is equal to j.

Proof. In case $f(i) = g(h(i))$ and $f(j) = g(h(j))$ are both defined and are the same, then $h(i)$ and $h(j)$ must both be defined and the same because g is strictly increasing and hence one-to-one. Consequently, $h(i)$ and $h(j)$ are indices of identical strings σ_i and σ_j, hence i and j must be the same. ◀

▶ Claim 2. For all $i \in I_s$, either $f(i)$ is undefined or $s < f(i)$.

Proof. Fix i in I_s. In case $h(i)$ is defined, by definition of h we have $\sigma_i = \tau_{h(i)}$, hence $h(i) > s$ by definition of I_s. Then also $f(i) = g(h(i)) > s$ because g is strictly increasing. ◀

▶ Claim 3. For all i it holds that $f(i) < f(f(i))$ whenever both values are defined.

Proof. It suffices to show that $h(i)$ is strictly less than $h(g(h(i)))$ because g is strictly increasing and maps these two indices to $f(i)$ and $f(f(i))$, respectively. In case the string $\sigma_{g(h(i))}$ occurs among $\tau_0, \tau_1, \ldots$ at all, then the corresponding index $h(g(h(i)))$ must be strictly larger than $h(i)$ because by choice of g, the string $\sigma_{g(h(i))}$ differs from τ_0 through $\tau_{h(i)}$. ◀

For every i there is a maximum natural number m such that $f^{[m]}(i)$ is defined because the strings $\sigma_{f^{[0]}(i)}, \sigma_{f^{[1]}(i)}, \ldots$ are mutually distinct and have all length at most $|\sigma_i|$. Given i and such maximum m, we let the i-CASCADE be the sequence

$$i, f(i), f(f(i)), \ldots, f^{[m]}(i)$$

and we call i, $f^{[m]}(i)$, and m respectively the STARTING POINT, the END POINT, and the LENGTH of this cascade. The minimum possible length of a cascade is 0, in which case starting point and end point coincide. Note that by choice of f, the length of an i-cascade can be equivalently defined as the least k such that $\sigma_{f^{[k]}(i)}$ is not b-compressible, i.e., an index j occurring in a cascade is the end point of the cascade if and only if σ_j is not in Γ^b_U, or equivalently, is in $\Gamma^{a\setminus b}_U$.

▶ Claim 4. If two cascades have the same end point, then the starting point of one cascade occurs in the other.

Proof. For a proof, consider an i-cascade of length k and a j-cascade of length $l \leq k$ that have the same end point. In case the j-cascade has length 0, there is nothing to prove. Otherwise, since f is one-to-one, the indices $f^{[l-1]}(i)$ and $f^{[k-1]}(i)$ must be the same, and by an easy induction argument we obtain

$$i = f^{[0]}(i) = f^{[k-l]}(j)\ . \qquad \blacktriangleleft$$

▶ Claim 5. Any two distinct starting points which belong to I_s have distinct end points for their respective cascades.

Proof. By Claims 2 and 3, the numbers that occur in a cascade that starts at any point in I_s are all strictly larger than s, except for the starting point, which has size at most s. So given two distinct indices $i, j \in I_s$, i cannot occur in the j-cascade and vice versa, hence the cascades starting at i and at j must have distinct end points by Claim 4. $\blacktriangleleft$

Let E be the set of all indices i that are end points of a cascade starting at some index in I^+_s. The cardinality of E is then equal to the cardinality of either I_s or I^+_s since by Claim 5 the end points of the cascades starting at indices in I^+_s are mutually distinct except that there may be a unique index $j \in I_s$ such that the j-cascade and the t-cascade have the same end point. In the latter case, the index t is equal to $f^{[k]}(j)$ for some $k > 0$ by $s < t$ and Claims 2, 3, and 4, hence the length of σ_j is at least $|\sigma_t| + 1$. Furthermore, in case there is such an index j, we have

$$\sum_{i\in E} 2^{-|\sigma_i|} \geq \sum_{i\in I^+_s\setminus\{j\}} 2^{-|\sigma_i|} \geq \Theta^{a\setminus b}_U + 3\cdot 2^{-n} - 2^{-|\sigma_j|} > \Theta^{a\setminus b}_U,$$

where the inequalities hold, first, by choice of the index j and because the strings indexed by a cascade decrease in length, hence increase in weight, second, by (5) and, third, because of $|\sigma_j| \geq |\sigma_t| + 1 = n - 1$. Otherwise, in case the end points of the cascades starting at some index in I^+_s are mutually distinct, we can argue similarly and infer rather directly from (5) that the weight of the strings with index in E is strictly larger than $\Theta^{a\setminus b}_U$. So we obtain in both cases a contradiction to the definition of $\Theta^{a\setminus b}_U$ because the end point of any cascade is the index of a string that is in Γ^a_U but not in Γ^b_U, hence this string contributes its weight to $\Theta^{a\setminus b}_U$. This concludes the proof of Theorem 11. $\blacktriangleleft$

6 Universal Theta numbers and Omega numbers

Finally, we ask which reals can be realized as one-sided or two-sided Theta numbers. Tadaki [9] demonstrates that one-sided universal Theta numbers are always Omega numbers. He then asks whether conversely every Omega number can be realized as one-sided universal Theta number. Similarly, by Theorems 10 and 11, which assert that any two-sided universal Theta number is indeed an Omega number in case the corresponding larger compression bound b is sufficiently large, it is suggesting to ask whether all Omega numbers can be realized as two-sided universal Theta numbers. We give a positive answer to both question in Theorem 12. Together with the results mentioned above this yields a new characterization of the Omega numbers: a real is an Omega number if and only if the real is a one-sided universal Theta number.

▶ **Theorem 12.** *Let a and $b > a$ be natural numbers and let α be a nonnegative left-r.e. Martin-Löf random real where $\alpha < 2^{-a}$. Then there are universal prefix-free machines V and V' such that $\alpha = \Theta_V^{a\backslash b} = \Theta_{V'}^{a}$.*

Acknowledgements The authors are grateful to anonymous referees of the STACS conference for pointing out an erroneous statement of one of the results and for their comments and corrections in general.

References

1 Cristian S. Calude, Peter H. Hertling, Bakhadyr Khoussainov, and Yongge Wang. Recursively enumerable reals and Chaitin Ω numbers. *Theoretical Computer Science*, 255(1-2):125–149, 2001.
2 Gregory J. Chaitin. A theory of program size formally identical to information theory. *Journal of the ACM*, 22:329–340, 1975.
3 Gregory J. Chaitin. Incompleteness theorems for random reals. *Advances in Applied Mathematics*, 8(2):119–146, 1987.
4 Rodney G. Downey and Denis R. Hirschfeldt. *Algorithmic randomness and complexity*. Springer, New York, 2010.
5 Antonín Kučera and Theodore A. Slaman. Randomness and recursive enumerability. *SIAM Journal on Computing*, 31(1):199–211, 2001.
6 Ming Li and Paul Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer, New York, third edition, 2008.
7 Joseph S. Miller and Liang Yu. Oscillation in the initial segment complexity of random reals. *Advances in Mathematics*, 226(6):4816–4840, 2011.
8 Robert I. Soare. *Recursively enumerable sets and degrees*. Springer-Verlag, Berlin, 1987.
9 Kohtaro Tadaki. A new representation of Chaitin Ω number based on compressible strings. In Cristian Calude, Masami Hagiya, Kenichi Morita, Grzegorz Rozenberg, and Jon Timmis, editors, *Unconventional Computation*, volume 6079 of *Lecture Notes in Computer Science*, pages 127–139. Springer, Berlin Heidelberg, 2010.
10 Alexander K. Zvonkin and Leonid A. Levin. The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematical Surveys*, 25(6):83–124, 1970.

Contraction checking in graphs on surfaces

Marcin Kamiński*[1] and Dimitrios M. Thilikos†[2]

1 **Département d'Informatique**
Université Libre de Bruxelles
Marcin.Kaminski@ulb.ac.be
2 **Department of Mathematics**
National and Kapodistrian University of Athens
sedthilk@math.uoa.gr

Abstract

The Contraction Checking problem asks, given two graphs H and G as input, whether H can be obtained from G by a sequence of edge contractions. Contraction Checking remains NP-complete, even when H is fixed. We show that this is not the case when G is embeddable in a surface of fixed Euler genus. In particular, we give an algorithm that solves Contraction Checking in $f(h, g) \cdot |V(G)|^3$ steps, where h is the size of H and g is the Euler genus of the input graph G.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases Surfaces, Topological Minors, Contractions, Parameterized algorithms, Linkages

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.182

1 Introduction

We consider simple finite graphs and use standard graph-theoretical terminology. For notions not define here, we refer the reader to Diestel [6] and to Mohar and Thomassen [17].

Contractions and topological minors. To *contract an edge* is to identify its two endpoints and remove the loop and multiple edges that have possibly been created. A graph H is a *contraction* of a graph G ($H <_c G$) if H can be obtained from G by a sequence of edge contractions. Deciding whether the input graph can be contracted to a fixed pattern is NP-complete, even for small pattern graphs – the smallest is an induced path on four vertices [3].

To *dissolve a vertex* of degree 2 is to contract one of the edges incident with it. A graph H is a *topological minor* of a graph G if H can be obtained from G by a sequence of vertex/edge deletions and vertex dissolutions. Recently, Grohe et al. proved that for every fixed graph H there exists an $\mathcal{O}(|V(G)|^3)$ time algorithm deciding whether H is a topological minor of G [13]. This is an FPT algorithm for this problem when parameterized by the size of H, that is, an algorithm with running time $g(|H|) \cdot |G|^{\mathcal{O}(1)}$. (For more information on parametrized complexity theory, see any of the books: Downey and Fellow [7], Flum and Grohe [10], or Niedermeier [20].)

Previous work on contractions. The problem of checking whether a graph is a contraction of another has attracted some attention. Perhaps the first systematic study of contractions

* Chargé de Recherches du FRS-FNRS.

† Supported by the project "Kapodistrias" (**ΑΠ** 02839/28.07.2008) of the National and Kapodistrian University of Athens (project code: 70/4/8757).

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 182–193

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

was undertaken by Brouwer and Veldman [3]. According to the results of [3], checking if a graph is contractible to the induced cycle on four vertices or the induced path on four vertices is NP-complete. More generally, they prove that it is NP-complete for every bipartite graph with at least one connected component that is not a star. Looking at contractions to fixed pattern graphs is justified by the result by Matoušek and Thomas [15] who proved that deciding, given two input graphs G and H, whether G is contractible to H is NP-complete even when both G and H are trees.

Surface containment relations. Surface versions of contractions and topological minors can be defined for surface-embedded graphs. Formal definitions are presented in Section 2. For the purpose of this introduction, we only note that surface contractions and surface topological minors are surface-embedded versions of contractions and topological minors, respectively, that respect the embedding.

For every surface Σ and every pattern graph H, there exists a polynomial-time algorithm deciding whether a Σ-embedded graph can be contracted to H [14]. The algorithm is based on a combinatorial lemma that allows to reduce the problem of testing for contraction in a surface-embedded graph to a constant number of tests for surface topological minors in its dual. The procedure is polynomial for every fixed graph H; however, the degree of the polynomial depends on the size of H. Is it possible to design an FPT algorithm for this problem when parameterized by the size of H?

The main obstacle is testing for surface topological minors. If there existed an FPT algorithm for deciding if a surface-embedded input graph contains a pattern graph H as a surface topological minor, then the machinery of [14] would imply an FPT algorithm for contraction checking. Surface topological minors are different from topological minors as they are defined for surface-embedded graphs and respect the embedding. While it is possible to reduce topological minor testing to surface topological minor testing, the latter is not known to be FPT-reducible to the former.

In this paper we overcome these difficulties and show that testing whether a surface-embedded graph is contractible to a given pattern is FPT, when parameterized by the size of the pattern.

The irrelevant vertex technique. A core technique from Graph Minors by Robertson and Seymour that has been especially prolific in algorithmic research is the following win/win approach. If the treewitdh of the input graph is small (less than a certain constant c depending on the problem parameter), apply dynamic programming and solve the problem in FPT time with respect to c; otherwise, exploit the existence of a subdivision of a large wall in the input graph (its size depends on c). In the latter case, one can usually find an *irrelevant vertex* – a vertex that can be safely removed from the graph without changing the solution. Then, the algorithm is recursively applied to the new graph so that, eventually, the treewidth of the graph drops below c to make the dynamic programming approach applicable.

Our approach. We follow this general scheme, however, we additionally prove that one can assume that the subgraph containing a large subdivided wall is of bounded treewidth. More precisely, for every positive integer h and a surface Σ, there exist constants t and T such that in every Σ-embedded graph of treewidth at least t there exists a disk in Σ such that the graph induced by the vertices inside the disk is of treewidth at most T and contains a subdivision of a wall of height h. This assumption comes in handy in our proof. We also believe that this lemma is of independent interest and can be applied to other problems.

Having found a subgraph of bounded treewidth containing a large subdivided wall, we consider a collection of nested cycles from the wall. For each cycle, we check what subpatterns of the guest graph can be seen as surface topological minors of its interior with a

Relation	planar graph	graphs on surfaces	all graphs
(induced) subgraph	FPT [8]		W[1]-hard
minor	FPT[22]		
topological minor	FPT [13]		
weak/strong immersion	FPT [13]		
induced minor	FPT [9]	OPEN	para-NP-complete [9]
contraction	FPT [this paper]		para-NP-complete [3]

Table 1 Overview of parameterized complexity status of containment relations in graphs.

"certain attachment" to the boundary of the cycle. This attachment determines the possible ways such a pattern should be extended outside the cycle towards matching the structure of the host graph. This is encoded as a *characteristic* function of each cycle. A key property is that the characteristic function is *monotone* – whatever can be attached to a cycle, can also be attached to subsequent cycles in the collection.

The main idea is to determine a collection of consecutive cycles with the same characteristic function, which is now feasible since this computation takes place in a graph of bounded treewidth. If this collection is "sufficiently large" then the monotonicity property implies that every sub-pattern of the guest graph can be also located away from some "safe" cycle and this is proved by making use of the Unique Linkage Theorem of Robertson and Seymour from [21, 23]. Then the safe cycle contains an irrelevant vertex that is removed and the procedure recurses until the host graph has bounded treewidth.

Table 1 summarizes the current state of research on parameterized complexity of containment relations, including the contribution of this paper.

In this extended abstract we give a detailed outline of the algorithm. The complete presentation of the algorithm and the proof will appear in the journal version of this paper.

2 Definitions

Surfaces. A *surface* Σ is a compact 2-manifold without boundary (we always consider connected surfaces). Whenever we refer to a Σ-*embedded graph* G we consider G accompanied by some embedding of it in Σ without crossings. To simplify notation, we do not distinguish between a vertex of G and the point of Σ used in the drawing to represent the vertex or between an edge and the line representing it. Given an edge e, we denote by $\overline{e}$ the set of its endpoints (clearly, $1 \leq |\overline{e}| \leq 2$). We also consider a graph G embedded in Σ as the union of the points corresponding to its vertices and edges. That way, a subgraph H of G can be seen as a graph H, where $H \subseteq G$. We refer to the book of Mohar and Thomassen [19] for more details on graph embeddings. The *Euler genus* of a graph G is the minimum integer γ such that G can be embedded on a surface of the Euler genus γ.

Given a Σ-embedded graph G, we denote by $F(G)$ the set of its faces, i.e. the set of connected components of the set $\Sigma \setminus G$. We say that a face in $F(G)$ is *trivial* if it is incident with at most two edges. An edge is *trivial* if it is incident with a trivial face. A loop of G is an edge with one endpoint. We say that a loop e is *singular* if it is either non-contractible or it is contractible and both connected components of $\Sigma \setminus e$ contain vertices of G.

The *surface contraction of an edge* e in a Σ-embedded graph G is the graph $G' = G\backslash_{\Sigma} e$ defined as follows. In case e is non-singular, G' is the graph obtained if we identify the closure of all points of e to a single vertex. In case e is singular the G' is the graph obtained

from G after removing all points of e. Notice that surface contractions are defined in a way that preserves surface integrity.

Let H and G be two Σ-embedded graphs. We say that H is a *surface contraction* of G, denoted by $H \leq_{\mathrm{c}}^{\Sigma} G$, if H can be obtained from G by a (possibly empty) sequence of operations that may be either surface contractions of edges or removals of trivial edges. Finally, we say that H is a *surface minor* of G, if H is a surface contraction of some subgraph of G.

Isomorphism. Let A_1 and A_2 be graphs and let $\psi\colon V(A_1) \to V(A_2)$ be a bijection. We say that A_1 and A_2 are *ψ-isomorphic* if for each pair $x, y \in V(A_1)$ it holds that $\{x, y\} \in E(A_1)$ if and only if $\{\psi(x), \phi(y)\} \in E(A_2)$. The *edge extension* of ψ, denoted by $\psi^e : V(A_1) \cup E(A_1) \to V(A_2) \cup E(A_2)$ extends ψ so to incorporate the correspondence between the edges of A_1 and the edges of A_2 implied by ψ.

Topological isomorphism. Let A_i be Σ_i-embedded graphs $i \in \{1, 2\}$. Suppose also that Σ_1 is homeomorphic to Σ_2. Let $\psi : V(A_1) \to V(A_2)$ be a bijection from $V(A_1)$ to $V(A_2)$. We say that A_1 is ψ-topologically isomorphic to A_2 is there is a homeomorphism $\phi : \Sigma_1 \to \Sigma_2$ such that ψ is an isomorphism from A_1 to A_2 and ψ^e is induced by the restriction of ϕ in $V(A_1)$. Notice that the bijection ψ above is an isomorphism between A_1 and A_2.

Surface topological minor. Let Σ be a surface and $\mathcal{G}$ be a Σ-embedded graph. Given a set $\mathcal{P}$ of internally disjoint extended paths of G, we define $G_{\mathcal{P}}$ as the Σ-embedded graph created if we first remove from G each edge not in a path in $\mathcal{P}$ and then replace each extended path (P, A) in $\mathcal{P}(G)$ by the extended path $((\overline{e}, \{e\}), A)$ where e is a new edge and $\overline{e} = A$.

Let Σ be a surface and (G, S_G) and (H, S_H) be two rooted Σ-embedded graphs. Let also σ be a bijection from S_G to S_H. We say that (H, S_H) is a *surface σ-rooted topological minor* of (G, S_G), and we denote it by $(H, S_H) \leq_{\sigma}^{\Sigma} (G, S_G)$ if there is a collection $\mathcal{P}$ of internally disjoint extended paths in G such that $G_{\mathcal{P}}$ is ψ-topologically isomorphic to H for some bijection $\psi : V(G_{\mathcal{P}}) \to V(H)$ where $\sigma \subseteq \psi$. When $S_G = S_H = \emptyset$, we say that H is a *surface topological minor* for G and denote it by $H \leq_{\mathrm{stm}}^{\Sigma} G$.

The main technical result of [14] is an equivalence between surface contractions in a surface-embedded graph and surface topological minors in its dual. A multigraph is called *thin* if it has no two parallel edges bounding a 2-face. (In particular, simple graphs are thin.) For a surface Σ and a simple Σ-embedded graph H, let $C_{\Sigma}(H)$ be a maximal set of thin Σ-embedded multigraphs that have the same adjacencies between their vertices as H (that is, forgetting multiple edges) such that they are all pairwise not topologically isomorphic. The set $C_{\Sigma}(H)$ is finite (Lemma 5 in [14]).

▶ Proposition 1 ([14]). Let G and H be graphs. Suppose also that G is embedded in a surface Σ and let G^* be its dual. Then $H \leq_{\mathrm{c}}^{\Sigma} G$ if and only if there exists a graph $\hat{H} \in C_{\Sigma}(H)$ such that $\hat{H}^* \leq_{\mathrm{stm}}^{\Sigma} G^*$.

3 Description of the algorithm

Let G and H be the host and the guest graph respectively. We denote by n the number of vertices in G. Also, in order to maintain only one parameter during the description of the algorithm, we assume that $h = |E(H)| + |V(H)| + \mathbf{eg}(G)$, where $\mathbf{eg}(G)$ is the Euler genus of G. For simplicity, we will use the notation $O_h(n^{\alpha})$ instead of $f(h) \cdot n^{\alpha}$ where f is some computable function of h.

General framework. Following the idea of the irrelevant vertex technique, introduced by Robertson and Seymour in [22], our first step is to check whether the treewidth of G is at

most $f_0(H) + h + 1$ where $f_0 : \mathbb{N} \to \mathbb{N}$ is a suitable function of H. This can be done in $O_h(n)$ steps because of the results in [2]. If $\mathbf{tw}(G) < f_0(h) + h + 1$, then the problem can be solved by the dynamic programming algorithm of [1] in $O_h(n)$ steps (this also follows from Courcelle's theorem [4] and the fact that contraction checking is expressible in Monadic Second Order Logic). So we may assume that $\mathbf{tw}(G) \geq f_0(h) + h + 1$. Also using the algorithm in [18] we may consider that G is optimally 2-cell embedded in some surface Σ of Euler genus $\mathbf{eg}(G)$. Let G^* be the dual embedding of G in Σ. From [16], the treewidth of a Σ 2-cell embedded graph and the treewidth of its dual cannot differ more than $\mathbf{eg}(\Sigma) + 1$. Therefore $\mathbf{tw}(G^*) \geq f_0(h)$. From Proposition 1, H is a contraction of G if and only if for some Σ-embedded graph in $\hat{H} \in C_\Sigma(H)$ it holds that $\hat{H}^* \leq^\Sigma_{\text{stm}} G^*$. Recall that the size of each graph in $C_\Sigma(H)$ depends only on H and $\mathbf{eg}(G)$ and therefore is bounded by $f_1(h)$ for some function f_1.

Our goal is to give an $O_h(n^2)$ step procedure with the following specifications:

Procedure Irrelevant Edge Detection(G, Σ)
Input: a graph G' of treewidth at least $f_0(h)$ that is 2-cell embedded in a surface Σ of Euler genus $\leq h$.
Output: an edge $e' \in E(G')$ such that $G' \setminus e$ remains 2-cell embedded in Σ and for every Σ-embedded graph H' of size at most $f_1(h)$, it holds that

$$H' \leq^\Sigma_{\text{stm}} G' \Leftrightarrow H' \leq^\Sigma_{\text{stm}} G' \setminus e'.$$

Actually, function f_0 should be chosen to be "sufficiently big" so it is possible to find an irrelevant edge.

Let e^* be the output of Irrelevant Edge Detection(G^*, Σ). Using the proof of Proposition 1, we may find an edge $e^* \in E(G^*)$ such that if e^* it is the dual edge of $e \in E(G)$, then H is a contraction of G if and only if H is a contraction of G/e. That way we reduce, in $O_h(n^2)$ steps, the problem of checking whether $H \leq_c G$ to the problem whether $H \leq_c G_{\text{new}} = G/e$. Clearly, we may again check whether $\mathbf{tw}(G_{\text{new}}) < f_0(h) + h + 1$ and either solve the problem by dynamic programming or again apply the Irrelevant Edge Detection procedure on G_{new}. Since the new graph is always smaller than the previous, applying the same steps, the algorithm will stop and produce a correct solution. As this will occur in less than n repetitions, the whole algorithm will take $O_h(n^3)$ steps, as claimed.

Given the above framework, what remains is to describe how the Irrelevant Edge Detection procedure works.

Big walls of small treewidth. It follows from the results in [5, 11, 12] that every Σ-embeddable graph of big enough treewidth contains as a subgraph a subdivision of a wall of given height and width (where height and width are defined in the obvious way). Also, by the same results, we can assume that this subdivision is "flat in the surface" in the sense that its perimeter is a contractible cycle of the embedding (i.e. handles are outside the wall). An example of such a subdivided wall is depicted in Figure 1 (for simplicity, we do not depict the subdivision vertices). We need the following Lemma:

▶ **Lemma 1.** *There are functions t_1 and t_2 such that, for every κ, every graph G that is embedded in a surface Σ of Euler genus g and has treewidth at least $t_1(\kappa, g)$, contains a subgraph R such that*

- *R is the subdivision of a wall of height and width equal to k,*
- *R is drawn inside a closed disk Δ bounded by its perimeter, and*

- $\Delta \cap G$, *i.e. the part of the graph that lies inside the perimeter of* R, *has treewidth upper bounded by* $t_2(\kappa, g)$.

Also, such a graph R *can be computed in* $O_h(n^2)$ *steps.*

Proof. The following claim can easily be derived by Lemma 4 in [11].

Claim. Let G be a graph embedded in a surface Σ of Euler genus g and let i be a positive integer. If $\mathbf{tw}(G) \geq 48i(g+1)$, then G contains a subdivided wall R of height i and width i as a subgraph and R is drawn inside a closed disk Δ of Σ bounded by the perimeter of G'.

Let $t_1(\kappa, g) = 48\kappa(g+1)$ and $t_2(\kappa, g) = 48(\kappa+1)(g+1)$. Apply the following routine on G.

1. Let $G' := G$.
2. While $\mathbf{tw}(G') \geq t_2(\kappa, g)$ do
3. let $i = \kappa + 2$,
4. let R' be a subdivided wall of height i, as in the above claim, and
5. update G' to the subgraph of G' induced by the vertices in the *strict* interior of the perimeter of R'.
6. Output G'.

Notice that the output of the above routine has always treewidth at most $t_2(\kappa, g)$. If the above algorithm never enters the loop of lines 3–5, then $\mathbf{tw}(G') = \mathbf{tw}(G) \geq t_1(\kappa, g)$ and, because of the above claim for $i = k$, G contains the desired subdivided wall R of height k. If this is not the case, then because of the stripping of Line 5, G' (and thus G as well) contains a wall R of height $i - 2 = k$, as required. ◀

The third assertion of Lemma 1 is important for our algorithm, as it implies that all subgraphs of G that are inside the outer cycle have bounded treewith and therefore, for these graphs, it is possible to answer queries on (rooted) surface topological minor containment in $O_h(n)$ steps.

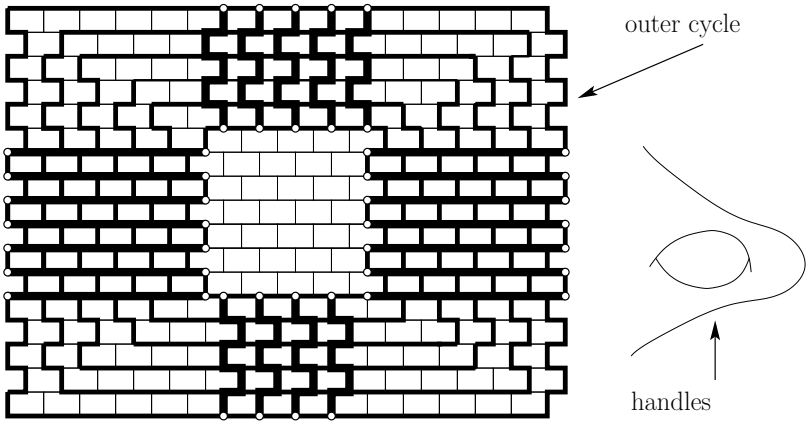


Figure 1 A wall of height 17 and width 15 together with a railed annulus of 6 cycles and 23 rails in it.

Cycles, rails, and tracks. Notice now that inside the perimeter of a subdivided wall of "big" enough height and width, one may distinguish a collection of nested cycles $\mathcal{A} = \{C_1, \ldots, C_r\}$

all met by a collection of paths $\mathcal{W} = \{W_1, \ldots, W_q\}$ (we call them *rails*) in a way that the intersection of a rail and a cycle is always a path. We can also assume that, among these cycles, C_r is the perimeter of the subdivided wall and we call it the *outer cycle*.

See Figure 1 for an example of how to extract 6 cycles and 23 rails from a (subdivided) wall of height 17 and width 15. We call this pair $(\mathcal{A}, \mathcal{W})$ of collections of cycles and rails *railed annulus* and observe that all rails and cycles are contained inside the outer cycle. Moreover, given that we need k_1 cycles and k_2 rails, we can always find them in a subdivided wall of big enough height and width. Combining this fact with Lemma 1, we derive the following.

▶ **Lemma 2.** *There exist functions t_3 and t_4, such that every graph G that is embedded in a surface Σ of Euler genus g and has treewidth at least $t_3(r, q)$ contains a* railed annulus *$(\mathcal{A}, \mathcal{W})$ if r cycles and q rails such that every subgraph of G that is entirely inside the outer cycle of $\mathcal{A}$ has treewidth at most $t_4(r, q)$.*

For a more abstract visualization of a railed annulus with 9 cycles and 24 rails, see Figure 2.

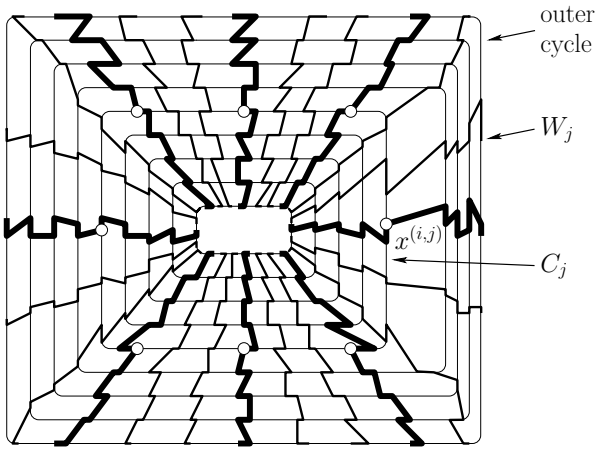


Figure 2 A railed annulus of 9 cycles and 24 rails. Among them, we distinguish 8 tracks.

For the purposes of our algorithm, we distinguish some proper subset of the rails and we call them *tracks*. For each cycle C_i of a railed annulus and for each rail W_h, we denote by $x^{(i,j)}$ the last vertex, starting from inside, of W_h that is a vertex of C_i. For the i-th cycle (counting from inside to outside) we denote by $X^{(i)}$ the set of all $x^{(i,j)}$'s on it (in Figure 2, $X^{(5)}$ consists of the white vertices). Also, for each i, we denote by $\Delta^{(i)}$ the inner closed disk bounded by C_i and by $G^{(i)}$ the subgraph of G that is is inside $\Delta^{(i)}$.

Crossings of a pattern graph. Let H be a Σ-embedded pattern graph of at most h edges and let $\tilde{\Delta}$ be a closed disk of Σ. The notion of a graph J that is $\tilde{\Delta}$-excised by H is visualized in Figure 5. Notice that J is embedded inside $\tilde{\Delta}$ and contains new vertices (the white vertices, denoted by X) that are the points of intersection of H with the boundary of $\tilde{\Delta}$. The number of these white vertices is the *crossing number* of J. We see each $\tilde{\Delta}$-excised graph J as being embedded inside the disk Δ. We also consider its *enhancement* $J_{\tilde{\Delta},X}$ by adding edges between boundary vertices as depicted in Figure 5. We say tha two $\tilde{\Delta}$-excised graphs J^1 and J^2 are *equivalent* if their enhancements $J^1_{\Delta,\mathrm{X}}$ and $J^2_{\Delta,\mathrm{X}}$ are topologically isomorphic.

We also define the same enhancement for each graph $G^{(i)}$ and we denote it by $G^{(i)}_{\Delta^{(i)},X^{(i)}}$ (see the left part of Figure 5).

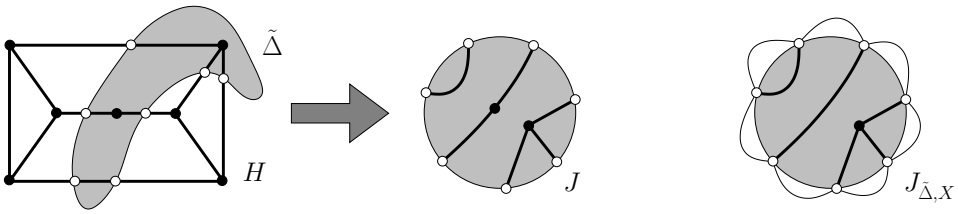


Figure 3 A graph J that is $\tilde{\Delta}$-excised by H and its enhanced version $J_{\tilde{\Delta},X}$ (X consists of the white vertices).

Attached topological minors. We set up a repository $\mathbf{H}_h$ of all graphs J that can be $\tilde{\Delta}$-excised by H with crossing number $f_4(h)$ where f_4 is a function to be determined later. Clearly, the size of $\mathbf{H}_h$ depends exclusively on h. Our next step is to set up a 0/1-vector χ_i that encodes, for every $J \in \mathbf{H}_h$ and every mapping $\rho : X \to X^{(i)}$, whether $J_{\tilde{\Delta},X}$ is a surface topological minor of $G^{(i)}_{\Delta^{(i)},X^{(i)}}$, where the vertices of X are mapped to vertices of $X^{(i)}$ as indicated by ρ. When this happens, we say that J is a *ρ-attached topological minor of $G^{(i)}$*). For an example of such a mapping, see the right part of Figure 5.

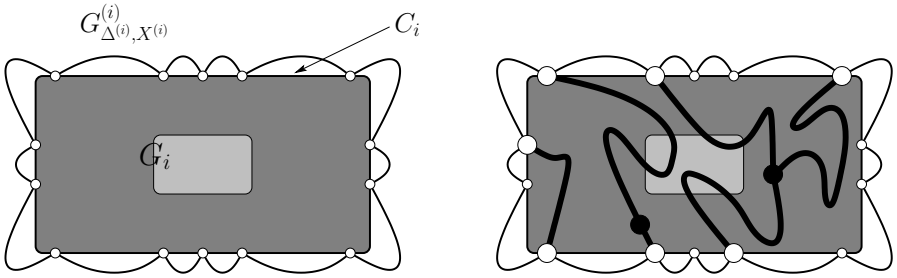


Figure 4 The graph $G^{(i)}_{\Delta^{(i)},X^{(i)}}$ and a realization of J as a ρ-attached topological minor of $G^{(i)}$.

Detecting an irrelevant edge. As each $G^{(i)}$ has bounded treewidth and the property of being a ρ-attached topological minor can be expressed in MSOL, χ_i can be computed in $O_h(n)$ steps and can be encoded in space that depends exclusively on h. It is important to notice that the vector sequence $\chi_1, \ldots, \chi_r$ is *monotone* in the sense that if a graph J is a ρ-attached topological minor of G^i, then it is also a ρ-attached topological minor of $G^{i'}$ for $i' > i$. By a pigeonhole argument, if the number of the cycles in the railed annulus is big enough, then there should exist a sub-collection $C_{\theta+1}, \ldots, C_{\theta+l}$ of consecutive cycles where $\chi_{\theta+1} = \ldots = \chi_{\theta+l}$, i.e., where the members of $\mathbf{H}_r$ behave the same as ρ-attached topological minors in their interiors (here l will be chosen to be as big as required for the correctness of our proofs). We call the seqence $C_{\theta+1}, \ldots, C_{\theta+l}$ *frozen* and observe that it can be detected algorithmically in $O_h(n)$ steps. In other words, we have the following:

▶ **Lemma 3.** *There exists some function $g : \mathbb{N} \to \mathbb{N}$ such that for every two positive integers h and l, every Σ-embedded graph G with a (r, q)-*railed annulus $(\mathcal{A}, \mathcal{W})$ *where $r \geq g(h) \cdot l$, and every $I \subset \{1, \ldots, q\}$ there is an integer $\theta \in \{0, \ldots, r - l\}$, such that the sequence $\{\chi_1, \ldots, \chi_r\}$ contains a subsequence $\{\chi_{\theta+1}, \ldots, \chi_{\theta+l}\}$ of l consecutive equal vectors. Moreover, there is an algorithm that, given h, l, G, $(\mathcal{A}, \mathcal{W})$, and I, outputs θ in $\phi(h, \mathbf{tw}(G^{(r)})) \cdot n$ steps, for some function ϕ.*

We claim that any edge in a non-track rail that lies between C_r and C_{r+1} is an irrelevant edge. In other words, the procedure Procedure Irrelevant Edge Detection(G, Σ) is the following:

Procedure Irrelevant Edge Detection(G, Σ)
1. Compute $\mathbf{H}_h$.
2. Find, using Lemma 2, a railed annulus $(\mathcal{A}, \mathcal{W})$
   in G with $r = g(h) \cdot t_3(h)$ cycles and $t_4(h)$ rails.
3. Pick a proper subset I of $\{1, \ldots, q\}$ of size $t_5(h)$
   and call the rails in $\{W_i \mid i \in I\}$ *tracks*.
4. Apply Lemma 3, using $(\mathcal{A}, \mathcal{W})$ and its tracks, in order to
   detect a frozen sequence $C_{\theta+1}, \ldots, C_{\theta+l}$ in $\mathcal{A}$.
6. Let $i \in \{1, \ldots .r\} \setminus I$ and let e be an edge of W_i that lies
   between $C_{\theta+1}$ and $C_{\theta+2}$, i.e. an edge in $W_i \cap (\Delta_{\theta+2} \setminus C_{\theta+1} \setminus \Delta_{\theta+2})$.
7. Output e.

The functions t_3, t_4, and t_5 above, depend on H and the genus of G and will be determined later so that the algorithm is correct.

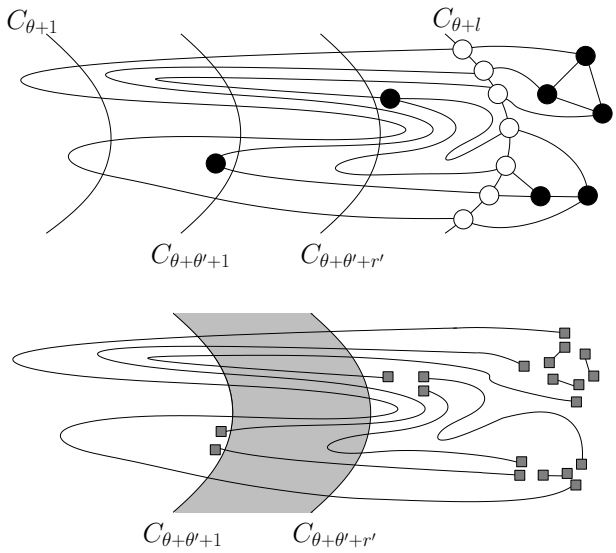


Figure 5 The upper figure depicts a realization of H as a topological minor of G. The annulus defined by the cycles $C_{\theta+\theta'+1}$ and $C_{\theta+\theta'+r'}$ does not contain any image of a vertex in H. The lower figure shows the corresponding linkage.

4 Correctness of the algorithm

This section contains a sketch of the proof that irrelevant edges are indeed irrelevant.

Linkage extraction. Suppose that H is a surface topological minor of G. Our purpose is to find a realization of H as a surface topological minor of G in a way that avoids the irrelevant edge. For this we fix our attention in the "frozen" annulus defined by the cycles $C_{\theta+1}$ and $C_{\theta+l}$. As H has at most $2 \cdot h$ vertices, there should be a big enough sub-annulus that does not contain any images of the vertices of H. Assume that this sub-annulus contains the

r' cycles $C_{\theta+\theta'+1}, \ldots, C_{\theta+\theta'+r'}$. Notice that H defines a collection of disjoint paths whose terminals are outside this annulus. This collection is a h'-*linkage* (i.e. a subgraph consisting of a collection of at most h' disjoint paths) for some $h' \leq h$ and we denote it by $\mathcal{L}'$ (see Figure 5).

Linkage replacement. The terminals of a linkage are the endpoints of its paths. Recall that the terminals of the linkage $\mathcal{L}$ that we detected in the previous paragraph has all its linkages outside the closed annulus defined by the cycles C_1 and C_r. We call such a linkage $\mathcal{A}$-avoiding linkage. Our next step is to prove the following lemma:

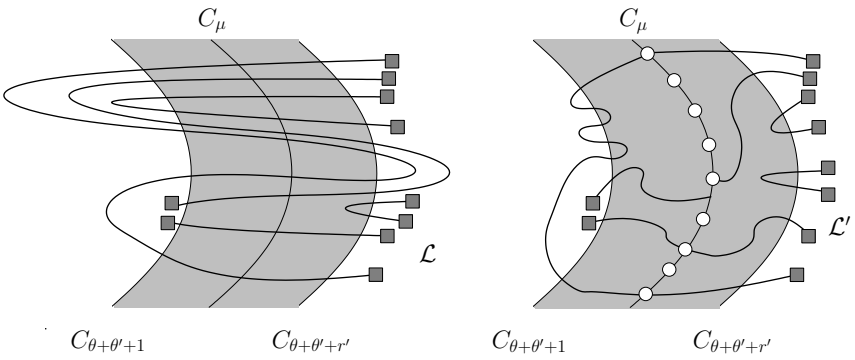


Figure 6 The replacement of linkage $\mathcal{L}$ by a linkage $\mathcal{L}'$. (We do not depict paths that are entirely outside the sub-annulus. Also, for reasons of simplicity we represent the intersection of all, except from one, paths with C_μ by a single vertex instead of a path.)

▶ **Lemma 4.** *There exist functions t_3, t_4, and t_5 such that the following hold: If h is a positive integer, G a Sigma-embedded graph with a railed annulus $(\mathcal{A}, \mathcal{W})$ with $r = t_3(h)$ cycles and $q = t_4(h)$ rails, $\mathcal{L}$ an $\mathcal{A}$-avoiding linkage $\mathcal{L}$ and subset I a proper subset of $\{1, \ldots, q\}$ where $|I| = t_5(h)$, then there is an $\mathcal{A}$-avoiding linkage $\mathcal{L}$ with the following properties:*

- *the paths of $\mathcal{L}$ link the same terminals as the paths in $\mathcal{L}'$,*
- *no more than $t_5(h)$ paths in $\mathcal{L}'$ cross the "middle" cycle $C_{\lceil r/2 \rceil}$ and, when this happens, their intersection will be just a path,*
- *when we orient such a path from inside to outside, its last in C_μ should always be a vertex of $X^{(\mu)}$.*

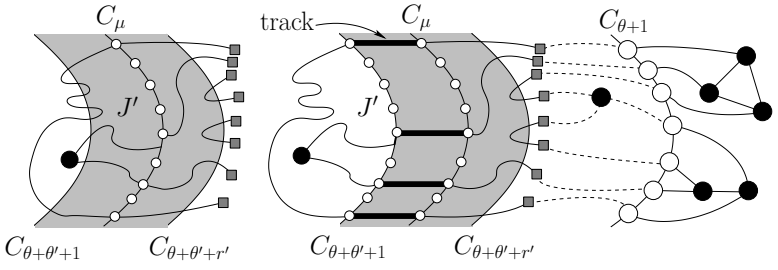


Figure 7 Two different realizations of J' as ρ-attached topological minors of G^μ. The one on the right avoids the irrelevant edge.

The proof of the above lemma is quite technical and uses the "vital linkage" Theorem of

Roberstong and Seymour in [23] (actually the function t_5 is directly taken from [23]). An example of this linkage replacement is depicted in Figure 6.

Pattern displacement. Our next step is to observe that the new linkage gives rise to a graph J' of $\mathbf{H}_h$ that is a ρ-attached topological minor of $G^{(\mu)}$. Recall that $\chi_{\theta+\theta'+1} = \chi_\mu$. Therefore, J' is also ρ'-attached topological minor of $G^{(\theta+\theta'+1)}$ where ρ' is the "left-side displacement" of ρ from C_μ to $C_{\theta+\theta'+1}$. But then, we may use the segments of the tracks that are cropped by the annulus defined by C_μ and $C_{\theta+\theta'+1}$ to realize J' as a ρ'-attached topological minor of $G^{(\mu)}$ in a way that rails that are not tracks are avoided (see Figure 7).

Clearly, the new realization of J' avoids the irrelevant edge and can be extended to a realization of H as a surface topological minor of G (see the right part of Figure 7). This means that the irrelevant edge is indeed irrelevant and this yields the correctness of procedure Irrelevant Edge Detection(G, Σ).

5 Open problem

We prove that contraction checking is FPT for graphs on surfaces. To complete Table 1 it would be interesting to know the parametrized complexity of induced minor checking for graphs on surfaces.

Acknowledgments

We wish to thank the anonymous reviewers of previous versions of this paper. Their comments and remarks were helpful in improving the presentation.

References

1 Isolde Adler, Frederic Dorn, Fedor V. Fomin, Ignasi Sau, and Dimitrios M. Thilikos. Fast minor testing in planar graphs. In *Algorithms - ESA 2010, 18th Annual European Symposium (1)*, volume 6346 of *Lecture Notes in Computer Science*, pages 97–109. Springer, 2010.

2 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

3 A. E. Brouwer and H. J. Veldman. Contractibility and NP-completeness. *Journal of Graph Theory*, 11(1):71–79, 1987.

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

5 Erik D. Demaine, Fedor V. Fomin, Mohammadtaghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. Assoc. Comput. Mach.*, 52(6):866–893, 2005.

6 Reinhard Diestel. *Graph Theory*. Springer-Verlag, Electronic Edition, 2005.

7 R.G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

8 David Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3), 1999.

9 Michael R. Fellows, Jan Kratochvíl, Martin Middendorf, and Frank Pfeiffer. The complexity of induced minors and related problems. *Algorithmica*, 13(3):266–282, 1995.

10 J. Flum and M. Grohe. *Parameterized complexity theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

11 Fedor V. Fomin, Petr A. Golovach, and Dimitrios M. Thilikos. Contraction obstructions for treewidth. *J. Comb. Theory, Ser. B*, 101(5):302–314, 2011.

12 J. F. Geelen, R. B. Richter, and G. Salazar. Embedding grids in surfaces. *European J. Combin.*, 25(6):785–792, 2004.

13 Martin Grohe, Ken-ichi Kawarabayashi, Dániel Marx, and Paul Wollan. Finding topological subgraphs is fixed-parameter tractable. *To appear in Proceedings of the 43rd ACM Symposium on Theory of Computing(STOC'11)*, 2011.

14 Marcin Kamiński, Daniël Paulusma, and Dimitrios M. Thilikos. Contractions of planar graphs in polynomial time. In Mark de Berg and Ulrich Meyer, editors, *ESA*, volume 6346 of *Lecture Notes in Computer Science*, pages 122–133. Springer, 2010.

15 Jirí Matousek and Robin Thomas. On the complexity of finding iso- and other morphisms for partial k-trees. *Discrete Mathematics*, 108(1-3):343–364, 1992.

16 Frédéric Mazoit. Tree-width of hypergraphs and surface duality. *CoRR*, abs/1006.3167, 2010.

17 B. Mohar and C. Thomassen. *Graphs on Surfaces*. The Johns Hopkins University Press, 2001.

18 Bojan Mohar. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM J. Discrete Math.*, 12(1):6–26, 1999.

19 Bojan Mohar and Carsten Thomassen. *Graphs on surfaces*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 2001.

20 Rolf Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

21 Neil Robertson and Paul D. Seymour. Graph Minors. XXII. Irrelevant vertices in linkage problems. *Journal of Combinatorial Theory, Series B*. (to appear).

22 Neil Robertson and Paul D. Seymour. Graph Minors XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

23 Neil Robertson and Paul D. Seymour. Graph Minors. XXI. Graphs with unique linkages. *Journal of Combinatorial Theory, Series B*, 99(3):583–616, 2009.

Distribution of the number of accessible states in a random deterministic automaton

Arnaud Carayol[1] **and Cyril Nicaud**[1]

1 **Université Paris-Est, LIGM, CNRS** {carayol,nicaud}@univ-mlv.fr

Abstract

We study the distribution of the number of accessible states in deterministic and complete automata with n states over a k-letters alphabet. We show that as n tends to infinity and for a fixed alphabet size, the distribution converges in law toward a Gaussian centered around $v_k n$ and of standard deviation equivalent to $\sigma_k\sqrt{n}$, for some explicit constants v_k and σ_k. Using this characterization, we give a simple algorithm for random uniform generation of accessible deterministic and complete automata of size n of expected complexity $O(n\sqrt{n})$, which matches the best methods known so far. Moreover, if we allow a ε variation around n in the size of the output automaton, our algorithm is the first solution of linear expected complexity. Finally we show how this work can be used to study accessible automata (which are difficult to apprehend from a combinatorial point of view) through the prism of the simpler deterministic and complete automata. As an example, we show how the average complexity in $O(n \log\log n)$ for Moore's minimization algorithm obtained by David for deterministic and complete automata can be extended to accessible automata.

1998 ACM Subject Classification F.2 Analysis of algorithms and problem complexity

Keywords and phrases finite automata, random sampling, average complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.194

1 Introduction

The structure of an automaton with n states over a k-letter alphabet is simply a deterministic and complete finite automaton with states in $[n] = \{1, \dots, n\}$ over the alphabet $\{a_1, \dots, a_k\}$. The state 1 is always assumed to be the unique initial state. We do not take final states into account as we are only interested in the structure of the automaton and not in the accepted language. We denote by $\mathcal{T}_{n,k}$ (or $\mathcal{T}_n$ if k is understood) the set of all such transition structures. As structures in $\mathcal{T}_{n,k}$ can alternatively be described by k-tuples of mappings from $[n]$ to $[n]$ (*i.e.* the i-th mapping corresponds to the action of the transitions labeled by a_i), the cardinal of $\mathcal{T}_{n,k}$ is $|\mathcal{T}_{n,k}| = n^{kn}$. An accessible automaton is a structure in $\mathcal{T}_{n,k}$ such that all states are accessible from the initial state 1. We denote by $\mathcal{A}_{n,k}$ (or $\mathcal{A}_n$ if k is understood) the set of all accessible automata in $\mathcal{T}_{n,k}$. The accessible automaton of a structure in $\mathcal{T}_{n,k}$ is obtained by restricting the structure to its

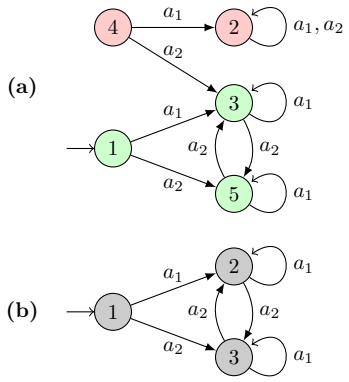


Figure 1 A structure **(a)** of $\mathcal{T}_{5,2}$ (a) and its accessible automaton **(b)** in $\mathcal{A}_{3,2}$.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 194–205
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

set $\{s_1 < \ldots < s_j\}$ of accessible states and by renaming the state s_i as i for all $i \in [j]$, as depicted in Fig. 1. Since $s_1 = 1$, the resulting automaton belongs to $\mathcal{A}_{j,k}$.

In this article, we study the distribution of the size of the accessible automaton of a random structure of $\mathcal{T}_{n,k}$, as n tends to infinity. The alphabet size $k \geq 2$ is assumed to be fixed and in particular does not depend on n (the case $k = 1$ is quite different and can be analyzed using known results on random mappings [9]). For all $n \geq 1$, we consider the random variable X_n describing the size of the accessible automaton of a structure in $\mathcal{T}_{n,k}$, for the uniform distribution. The probability for X_n to take value i, for $i \in [n]$, is given by the following formula first obtained in [16]:

$$P(X_n = i) = \frac{\overbrace{\binom{n-1}{i-1}}^{\text{labels of acc. states}} \cdot \overbrace{|\mathcal{A}_{i,k}|}^{\text{acc. aut.}} \cdot \overbrace{n^{k(n-i)}}^{\text{remaining transitions}}}{n^{kn}}. \tag{1}$$

Indeed if we fix the accessible automaton, it remains to choose the labels in $[n]$ for its states (as the initial state is always labeled by 1, we have $\binom{n-1}{i-1}$ choices) and the target for the $k(n-i)$ transitions that take their source outside of the accessible component (kn total transitions for the structure minus the ki of the accessible automaton). For these transitions, all n choices of target are valid. An important consequence of this formula is that two accessible automata in $\mathcal{A}_{i,k}$ appear in the same number of structures of $\mathcal{T}_{n,k}$, for any $i \in [n]$.

n	100	1000	10000
$\mathbb{E}[X_n]$(k=2)	79.6356	796.663	7967.41
$\mathbb{E}[X_n]$(k=3)	94.0138	940.489	9404.40
$\mathbb{E}[X_n]$(k=4)	97.9746	980.137	9801.89

k	2	3	4
v_k	0.796812	0.940479	0.980176

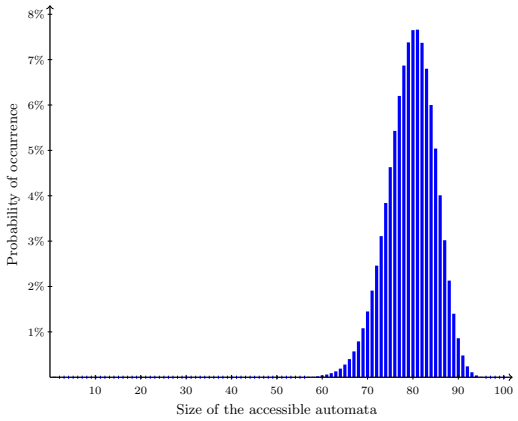


Figure 2 On the top left, an approximation of the average size of the accessible automaton based on 10000 randomly generated structures from $\mathcal{T}_{n,k}$. On the bottom left, the values of the constant $v_k = 1 + \frac{1}{k}W_0(-ke^{-k})$ for different values of k. On the right, the graphical representation of X_{100}.

Our main technical contribution is to describe the law of X_n for large values of n. As hinted by Fig. 2, we show that the average size of the accessible automaton $\mathbb{E}[X_n]$ is equivalent to $v_k n$ where v_k is a constant depending[1] on the size of the alphabet k. We also show that the standard deviation is equivalent to $\sigma_k\sqrt{n}$, where σ_k is also a constant depending on k. As shown in Fig. 2, the shape of the repartition of the size of the accessible automaton for a fixed n looks like a Gaussian. This type of behavior is quite common with combinatorial objects: it is the case for instance for the number of cycles in a random permutation of size n, for the number of occurrences of a fixed pattern in a random string of length n, ... (see

[1] Recall k is assumed to be fixed in our asymptotic analysis.

[10, p. 683]). It is formally captured by the notion of convergence in distribution to the normal (or Gaussian) law. More precisely, we are going to show that in a random structure of size n, once it has been centered by its mean and scaled by its standard deviation, the distribution of the size of the accessible automaton is asymptotically Gaussian. Note that standard analytic methods [10] cannot be directly applied here since there are no known expressions for the associated generating function.

Our interest in studying the distribution of the size of the accessible automaton is not only motivated by its fundamental nature but also by its rich implications in the algorithmic and combinatorial study of accessible automata. To substantiate our claim, we provide three applications of our theoretical results.

Our first application deals with the problem of uniform generation of deterministic and complete accessible automata. This problem is declined in two variants: the exact one and the ε-approximated one for $\varepsilon \in (0,1)$. The exact generation problem asks to generate uniformly at random an automaton of $\mathcal{A}_m$, for a given size $m \geq 1$, whereas the ε-approximated one asks for an automaton in $\mathcal{A}_{m'}$ for some $m' \in [(1-\varepsilon)m, (1+\varepsilon)m]$ where m is given; the ε-approximated also requires that two automata of the same size have the same probability to be generated. The first solution [17, 4] to the exact generation problem, based on an adaptation of the recursive method [18], has a complexity in $O(m^2)$ (consisting of a preprocessing in $O(m^2)$ and a computation in $O(m)$). In [1], another solution based on a representation of deterministic and complete accessible automata by words was proposed, with a complexity in $O(m^2)$. This complexity was later improved in [3], using methods based on combinatorial bijections and Boltzmann sampling [7], which gives an expected complexity of $O(m\sqrt{m})$. This last work was then adapted to generate possibly incomplete automata [2], with the same expected complexity. Note that the best known upper-bound for the ε-approximated problem is also $O(m\sqrt{m})$ as all known solutions to the ε-approximated problem are in fact solutions to the exact problem.

We propose a very simple algorithm for generating accessible automata of size m whose expected complexity $O(m\sqrt{m})$ matches the best known upper bounds. This algorithm consists in generating uniformly at random a transition structure in $\mathcal{T}_{n,k}$ with $n = \left\lfloor \frac{m}{v_k} \right\rfloor$ states and then to compute its accessible automaton. If it is of size m we output it, and otherwise we restart the process. The correctness of the algorithm follows from the above remark that two accessible automata of size m appear as accessible automata in the same number of structures of $\mathcal{T}_n$. The probability to obtain an accessible automaton of size exactly m is in $\Theta(\frac{1}{\sqrt{m}})$ and hence the average number of iterations of the algorithm is in $O(\sqrt{m})$. As every iteration can be computed in linear time, the expected complexity is in $O(m\sqrt{m})$.

Slightly modifying the algorithm to output the automaton when its size belongs to the interval $[(1-\varepsilon)m, (1+\varepsilon)m]$ yields a solution to the ε-approximation problem with an expected complexity of $O(m)$. We also show that this algorithm can readily be adapted to generate minimal automata (using a recent result on the asymptotic number of minimal automata [11]), with the same expected complexity for the exact version and an expected complexity in $O(n \log\log n)$ for the approximated version.

The second application concerns the formula expressing the asymptotic number of automata in $\mathcal{A}_{n,k}$, as n tends to infinity. In [14], Korshunov established that:

$$|\mathcal{A}_{n,k}| \sim E_k n! \left\{ {kn \atop n} \right\} \quad \text{with} \quad E_k = \frac{1 + \sum_{r=1}^{\infty} \frac{1}{r} \binom{kr}{r-1} (e^{k-1}\lambda_k)^{-r}}{1 + \sum_{r=1}^{\infty} \binom{kr}{r} (e^{k-1}\lambda_k)^{-r}} \text{ and } \lambda_k = \frac{e^{kv_k} - 1}{e^{k-1} v_k^k}, \quad (2)$$

where $\left\{ {kn \atop n} \right\}$ designates the Stirling numbers of the second kind: $\left\{ {kn \atop n} \right\}$ is the number of different

ways to partition kn elements into n non-empty sets. Recently in [15], Lebensztayn gave a simplified expression of the constant E_k, using the theory of Lagrange inversion applied in the context of generalized binomial series. Using our main result and the simple fact that $\sum_{i=1}^{n} P(X_n = i) = 1$, we obtain another proof of his simplified expression for E_k. Note that we do use Korshunov's equivalent to obtain our results but never the expression of the constant E_k given in Eq. (2).

The last application and the main perspective of this work is the study of combinatorial properties of accessible automata (which are difficult to apprehend from a combinatorial point of view) through the prism of the simpler structures of $\mathcal{T}_{n,k}$. This approach seems particularly well suited for the average case analysis of classical algorithms on finite automata. We give two examples of asymptotic properties of structures that can be transfered to accessible automata. In particular, we show that the average complexity in $O(n \log\log n)$ for Moore's minimization algorithm recently obtained for structures by David in [6] can be extended, using our result, to accessible automata.

2 Preliminaries

2.1 Deterministic and complete automata

A deterministic and complete transition structure for an automaton over a finite alphabet Γ is a tuple (Q, q_0, δ, F) where Q is a finite set of states, $q_0 \in Q$ is the initial state, $\delta : Q \times \Gamma \mapsto Q$ is the transition function and $F \subseteq Q$ is the set of final sets. For all $n \geq 1$ and $k \geq 2$, we denote by $\mathcal{T}_{n,k}$ the set of all structures over the alphabet $\{a_1, \ldots, a_k\}$ with states in $[n] = \{1, \ldots, n\}$, such that 1 is the initial state and with an empty set of final states.

A structure in $\mathcal{T}_{n,k}$ is said to be accessible (accessible automaton for short) if all its states can be reached from the initial state 1. We denote[2] by $\mathcal{A}_{n,k}$ the set of all accessible automata in $\mathcal{T}_{n,k}$. For a more detailed introduction to finite automata, we refer the reader to [13].

The equivalent of Korshunov given in Eq. (2) involves the Stirling numbers of the second kind $\left\{ {kn \atop n} \right\}$. In [12], Good establishes the following equivalent as n tends to infinity and for a fixed k:

$$\left\{ {kn \atop n} \right\} \sim \frac{(kn)!(e^{\rho_k} - 1)^n}{n! \rho_k^{kn} \sqrt{2\pi kn(1 - ke^{-\rho_k})}} \quad \text{with} \quad \rho_k = k + W_0(-ke^{-k}), \tag{3}$$

where the classical Lambert function W_0 [5] is implicitly defined by $W_0(x)e^{W_0(x)} = x$ and $W_0(x) \geq -1$ for all $x \geq -e^{-1}$. Alternatively, ρ_k is the unique positive solution of $\rho_k = k - ke^{-\rho_k}$.

Using Eq. (3) and Stirling's formula in Korshunov's equivalent (cf. Eq. (2)), we obtain:

$$|\mathcal{A}_{n,k}| \sim E_k \alpha_k \beta_k^n n^{kn} \quad \text{with } \alpha_k = \frac{1}{\sqrt{1 - ke^{-\rho_k}}} \text{ and } \beta_k = \frac{k^k(e^{\rho_k} - 1)}{\rho_k^k e^k}. \tag{4}$$

2.2 Elements of probability

Let us first recall some basic definitions of probability theory (see [8, 10] for more details). If X is a real valued random variable, we denote by $\mathbb{E}[X]$ its expected value and by $\mathbb{V}[X]$ its

[2] Instead of labeled automata, we could consider unlabeled automata: the set $\mathcal{A}^u_{n,k}$ of deterministic and complete automata with n states over $\{a_1, \ldots, a_k\}$ up to isomorphism. As deterministic and accessible automata do not admit non-trivial automorphisms, we have $|\mathcal{A}_{n,k}| = (n-1)!|\mathcal{A}^u_{n,k}|$. Remark that this property does not hold for non-accessible structures or non-deterministic automata.

variance, when they exist. The standard deviation is $\sqrt{\mathbb{V}[X]}$.

▶ **Definition 1.** Let $(X_n)_{n\geq 1}$ be a sequence of real valued random variables and X be a real valued random variable. We say that X_n *converges in distribution to* X when for every $t \in \mathbb{R}$, $P(X_n \leq t) \to P(X \leq t)$ as $n \to \infty$.

▶ **Definition 2.** Let $(X_n)_{n\geq 1}$ be a sequence of random variables such that $\mathbb{E}[X_n]$ and $\mathbb{V}[X_n]$ exist for all $n \geq 1$. We say that X_n *is asymptotically Gaussian* when the standardized random variable $X_n^* = \frac{X_n - \mathbb{E}[X_n]}{\sqrt{\mathbb{V}[X_n]}}$ converges in distribution to the normal distribution $\mathcal{N}(0,1)$ of parameters 0 and 1, defined by, for any $t \in \mathbb{R}$, $P\big(\mathcal{N}(0,1) \leq t\big) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{t} e^{-x^2/2} dx$.

3 Distribution of the size of the accessible component

In this section we state and prove our main result from which we derive all announced properties of this paper. This result, in particular, explains the Gaussian shape of Fig. 2.

▶ **Theorem 3** (Asymptotically Gaussian)**.** *Let X_n be the random variable associated with the size of the accessible part in a structure of $\mathcal{T}_{n,k}$. Then X_n is asymptotically Gaussian with expected value and standard deviation asymptotically equivalent to $v_k\, n$ and $\sigma_k\sqrt{n}$ respectively, with*

$$v_k = 1 + \frac{1}{k} W_0(-ke^{-k}) \quad \textit{and} \quad \sigma_k = \sqrt{\frac{v_k(1-v_k)}{kv_k - k + 1}}. \tag{5}$$

3.1 Outline of the proof of Theorem 3

In this section we present the ideas of the proof of Theorem 3. To shorten the presentation, we do not derive the asymptotic value of the expected value and variance before establishing the convergence in distribution to the Gaussian law. Initially, we estimated the values of the expected value and variance from Eq. (10) and Eq. (11) respectively. As shown in the next section, the proof of Theorem 3 can be reduced to the following statements: for all $\ell \in \{0,1,2\}$ and for all $t \in \mathbb{R}$, as n tends to infinity we have

$$\sum_{i=1}^{\lfloor v_k n\rfloor + \lfloor t\sqrt{n}\rfloor} \left(\frac{i - \lfloor v_k\, n\rfloor}{\sqrt{n}}\right)^{\ell} \cdot P(X_n = i) \;\longrightarrow\; \frac{1}{\sigma_k\sqrt{2\pi}} \int_{-\infty}^{t} x^{\ell} \cdot \exp\left(-\frac{x^2}{2\sigma_k^2}\right) dx, \tag{6}$$

$$\sum_{i=\lfloor v_k n\rfloor + \lfloor t\sqrt{n}\rfloor}^{n} \left(\frac{i - \lfloor v_k\, n\rfloor}{\sqrt{n}}\right)^{\ell} \cdot P(X_n = i) \;\longrightarrow\; \frac{1}{\sigma_k\sqrt{2\pi}} \int_{t}^{\infty} x^{\ell} \cdot \exp\left(-\frac{x^2}{2\sigma_k^2}\right) dx, \tag{7}$$

and there exists a positive constant $C > 0$ such that, for every i and n such that $1 \leq i \leq n$,

$$P(X_n = i) \leq \frac{C}{\sqrt{n}} \quad \text{and} \quad P(X_n = \lfloor v_k n\rfloor) \sim \frac{E_k \alpha_k \sqrt{v_k}}{\sqrt{2\pi n(1-v_k)}}. \tag{8}$$

3.1.1 Expected value and variance

Assuming that Eq. (6), Eq. (7) and Eq. (8) hold, we show how to establish Theorem 3.

For the expected value of X_n, we consider the sum of Eq. (6) and Eq. (7) for $\ell = 1$ and $t = 0$:

$$\frac{\mathbb{E}(X_n) - \lfloor v_k\, n\rfloor}{\sqrt{n}} = \sum_{i=1}^{n} \frac{i - \lfloor v_k\, n\rfloor}{\sqrt{n}} \cdot P(X_n = i) \longrightarrow \frac{1}{\sigma_k\sqrt{2\pi}} \int_{-\infty}^{\infty} \underbrace{x \cdot \exp\left(-\frac{x^2}{2\sigma_k^2}\right)}_{\text{odd function}} dx = 0$$

This proves that $\mathbb{E}[X_n] = \lfloor v_k\, n \rfloor + o(\sqrt{n}) = v_k\, n + o(\sqrt{n})$.

Similarly for the variance of X_n, we consider the sum of Eq. (6) and Eq. (7) for $\ell = 2$ and $t = 0$, we prove $\mathbb{E}\left[(X_n - \lfloor v_k\, n \rfloor)^2\right] \sim \sigma_k^2\, n$. It follows that $\mathbb{V}[X_n] \sim \sigma_k^2\, n$.

For the convergence in distribution to a normal distribution, we have, using the two equivalents obtained previously, that $P(X_n^* \leq t) \sim P(X_n \leq v_k\, n + \sigma_k\, t\sqrt{n})$. The error terms can be handled with Eq. (8), so that using Eq. (6) for $\ell = 0$ gives the result.

Note that we cannot deduce $\mathbb{E}[X_n] \sim v_k\, n$ and $\mathbb{V}[X_n] \sim \sigma_k^2\, n$ from the case $\ell = 0$ only, since the convergence in distribution of Y_n to Y does not necessarily imply that $\mathbb{E}[Y_n] \to \mathbb{E}[Y]$ or $\mathbb{V}[Y_n] \to \mathbb{V}[Y]$. In particular here, one can prove that not all the moments of X_n^* converge.

3.1.2 Reducing the range of the sums of Eq. (6) and Eq. (7)

Our first step of the proof is to show there exist two reals a and b such that $\frac{1}{e} < a < v_k < b < 1$ and

$$\sum_{i=1}^{\lfloor an \rfloor} P(X_n = i) + \sum_{i=\lfloor bn \rfloor}^{n} P(X_n = i) = o\left(\frac{1}{n}\right). \tag{9}$$

This is proved using classical upper bounds for binomial coefficients and for the number of automata in Eq. (1). As $\left(\frac{i - \lfloor v_k\, n \rfloor}{\sqrt{n}}\right)^\ell \in O(n^{\ell/2})$, this also shows that for $\ell \in \{1, 2\}$,

$$\sum_{i=1}^{\lfloor an \rfloor} \left(\frac{i - \lfloor v_k\, n \rfloor}{\sqrt{n}}\right)^\ell P(X_n = i) + \sum_{i=\lfloor bn \rfloor}^{n} \left(\frac{i - \lfloor v_k\, n \rfloor}{\sqrt{n}}\right)^\ell P(X_n = i) = o(1).$$

For the reminder of the proof, we fix a and b and we denote by I_n the set $\{\lfloor an \rfloor + 1, \ldots, \lfloor bn \rfloor\}$.

3.1.3 Equivalent of $P(X_n = i)$ for $i \in I_n$.

Our starting point is Eq. (1), which states that $P(X_n = i) = \frac{i}{n}\binom{n}{i}|\mathcal{A}_{i,k}|n^{-ki}$. For any integer i in I_n we can use the equivalent for $|\mathcal{A}_{i,k}|$ of Eq. (4) and Stirling's formula to obtain the following equivalent of $P(X_n = i)$:

$$P(X_n = i) \sim \frac{E_k \alpha_k}{\sqrt{2\pi n}}\, g\left(\frac{i}{n}\right)\left[f\left(\frac{i}{n}\right)\right]^n, \text{ with } f(x) = \frac{x^{(k-1)x}\beta_k^x}{(1-x)^{1-x}} \text{ and } g(x) = \sqrt{\frac{x}{1-x}}. \tag{10}$$

The constants α_k and β_k of Eq. (4) can be reformulated in terms of v_k using the facts that $v_k = \frac{\rho_k}{k}$ and $v_k = 1 - e^{-kv_k}$. We have $\alpha_k = (1 - ke^{-kv_k})^{-1/2}$ and $\beta_k = \frac{1}{(1-v_k)v_k^{k-1}e^k}$. As i belongs to I_n, $\frac{i}{n}$ belongs to $[a, b]$. On $[a, b]$ the function g is continuous and positive: it has little influence on the analysis. The situation is different for f because it is raised to the power n in the expression. When n grows, the distribution of probabilities is concentrated around the unique point v_k of $[a, b]$ where f reaches its maximum 1. The function f is positive on $[a, b]$, increasing on $[a, v_k]$ and decreasing on $[v_k, b]$, with $f(v_k) = 1$ and $f'(v_k) = 0$, as shown in Fig. 3. Notice that, since g is bounded on $[a, b]$ and

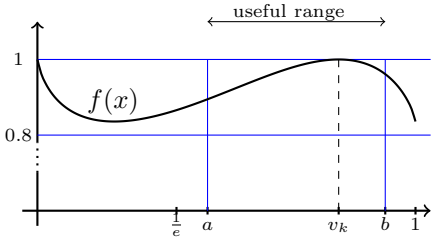


Figure 3 The variations of f on $[0, 1]$.

$|f| \leq 1$ on this interval, we have for all $i \in I_n$ that $P(X_n = i) \leq \frac{C}{\sqrt{n}}$ for some $C \geq 0$. This, and Eq (9) for values of i outside of I_n, proves the first part of Eq. (8).

We now set $i = \lfloor v_k n \rfloor + j$ to center around $\lfloor v_k n \rfloor$. Using Taylor's formula near v_k on $n \ln f(x)$ and remarking that $f''(v_k) = -\frac{1}{\sigma_k^2}$ we get:

$$f\left(\frac{\lfloor v_k n \rfloor + j}{n}\right)^n = \exp\left(-\frac{j^2}{2\sigma_k^2 n}\right)\left(1 + O\left(\frac{j^3}{n^2}\right) + O\left(\frac{1}{n}\right)\right). \tag{11}$$

This equation and Eq. (10) for $j = 0$ proves the second part of Eq. (8).

The function $x \mapsto e^{-x^2/2\sigma^2}$ appears in this formula, applied to $x = \frac{j}{\sqrt{n}}$, from which we will eventually obtain the asymptotic Gaussian shape. This hints that everything meaningful happens at scale $\sqrt{n}$ around $\lfloor v_k n \rfloor$. We now want to consider the sum where Eq. (11) is useful, that is, on a range where it contains every window of scale $\sqrt{n}$ and also where $\frac{j^3}{n^2}$ is not too big, for the Gaussian approximation to hold. For these reasons[3], we take a window of scale $n^{5/9}$ for j (we have $\sqrt{n} \ll n^{5/9} \ll n^{3/2}$). One can verify that the contribution outside of this window is negligible:

$$\sum_{i=\lfloor an \rfloor + 1}^{\lfloor v_k n \rfloor - \lfloor n^{5/9} \rfloor} P(X_n = i) + \sum_{i=\lfloor v_k n \rfloor + \lfloor n^{5/9} \rfloor}^{\lfloor bn \rfloor} P(X_n = i) = o\left(\frac{1}{n}\right). \tag{12}$$

For the first sum, we use that f is increasing on $[a, v_k]$, so that we can bound from above $P(X_n = i)$ by its value computed from the estimation of Eq. (11) with $i = v_k n - n^{5/9}$; this is enough to obtain the result. The second sum is calculated similarly.

3.1.4 Approximation by an integral at scale $\sqrt{n}$ around $v_k n$

At this point we have reduced the range of the sum to $\{\lfloor v_k n \rfloor - \lfloor n^{5/9} \rfloor, \ldots, \lfloor v_k n \rfloor + \lfloor n^{5/9} \rfloor\}$, and we aim at proving the following result: for all $t \in \mathbb{R}$,

$$\sum_{j=-\lfloor n^{5/9} \rfloor}^{\lfloor t\sqrt{n} \rfloor} \left(\frac{j}{\sqrt{n}}\right)^\ell P(X_n = \lfloor v_k n \rfloor + j) \xrightarrow[n\to\infty]{} \frac{E_k \alpha_k\, g(v_k)}{\sqrt{2\pi}} \int_{-\infty}^{t} x^\ell \cdot \exp\left(-\frac{x^2}{2\sigma_k^2}\right) dx. \tag{13}$$

In the working range of this section, we can use both Eq. (10) and Eq. (11). By Taylor's formula, $g(v_k + O(\frac{1}{n})) = g(v_k) + O(\frac{1}{n})$, and therefore, for all $j \in \{-\lfloor n^{5/9} \rfloor, \ldots, \lfloor n^{5/9} \rfloor\}$,

$$P(X_n = \lfloor v_k n \rfloor + j) = \frac{E_k \alpha_k\, g(v_k)}{\sqrt{2\pi n}} \exp\left(-\frac{j^2}{2\sigma_k^2 n}\right)\left(1 + O(n^{-1/3}) + O(\kappa_n)\right), \tag{14}$$

for some positive sequence $(\kappa_n)_{n\geq 1}$ that tends to 0 as n tends to infinity, which comes from Eq. (10): it is the maximum of the error term for $j \in I_n$.

Let h_ℓ be the function defined on $\mathbb{R}$ by $h_\ell(x) = x^\ell \cdot \exp\left(-\frac{x^2}{2\sigma_k^2}\right)$ and let $(\omega_n)_{n\geq 1}$ be a sequence of positive reals[4] with $\omega_n \to +\infty$, $\omega_n \cdot \kappa_n \to 0$ and $\omega_n \cdot n^{-1/9} \to 0$ as $n \to \infty$. Using this properties, one can obtain from Eq. (14) that for any fixed real t,

$$\sum_{j=-\lfloor \omega_n \sqrt{n} \rfloor}^{\lfloor t\sqrt{n} \rfloor} \left(\frac{j}{\sqrt{n}}\right)^\ell P(X_n = \lfloor v_k n \rfloor + j) = \frac{E_k \alpha_k\, g(v_k)}{\sqrt{2\pi n}} \sum_{j=-\lfloor \omega_n \sqrt{n} \rfloor}^{\lfloor t\sqrt{n} \rfloor} h_\ell\left(\frac{j}{\sqrt{n}}\right) + o(1).$$

---

[3] There are other technical reasons for which $n^{\frac{5}{9}}$ is a better choice than others n^λ with $\frac{1}{2} < \lambda < \frac{2}{3}$, but these are the main ones.

[4] For instance $\omega_n = \min\{\log n, -\log \kappa_n\}$ for n large enough.

Here we recognize a Riemann sum of step $\frac{1}{\sqrt{n}}$. Using that h'_ℓ is bounded on $\mathbb{R}$, we can therefore prove that it can be approximated by an integral (see Figure 4) as follows:

$$\frac{1}{\sqrt{n}} \sum_{j=-\lfloor \omega_n \sqrt{n} \rfloor}^{\lfloor t\sqrt{n} \rfloor} h_\ell\left(\frac{j}{\sqrt{n}}\right) = \int_{-\omega_n}^{t} h_\ell(x)dx + O\left(\frac{\omega_n}{\sqrt{n}}\right). \tag{15}$$

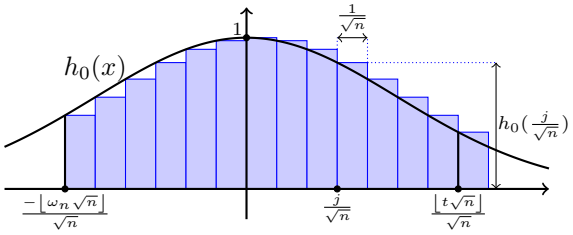


Figure 4 The sum is equal to the area in blue. It is well approximated, when n grows, by the surface below the curve between $-\omega_n$ and t, since the rectangles' width is smaller and smaller. For our function h_ℓ, the error term of this approximation is in $O(\frac{\omega_n}{\sqrt{n}})$, including the last rectangle.

It remains to estimate the sum for j in $\{-\lfloor n^{5/9} \rfloor, \ldots, \lfloor t\sqrt{n} \rfloor\}$. We use integral bounds, which are similar to Riemann sums, to obtain that there exists a constant $D > 0$ such that

$$\sum_{j=-\lfloor n^{5/9} \rfloor}^{-\lfloor \omega_n \sqrt{n} \rfloor} \left(\frac{j}{\sqrt{n}}\right)^\ell P(X_n = \lfloor v_k \rfloor\ n + j) \leq D \int_{-\infty}^{-\omega_n} |h_\ell(x)|dx.$$

Since $-\omega_n \to -\infty$, this part is asymptotically negligible, completing the proof of Eq. (13).

Hence, using Eq. (9) and Eq. (12) we obtain the proof that Eq. (6) holds. The same techniques can also be applied in order to prove Eq. (7).

3.1.5 Another proof of Lebensztayn's theorem [15]

Observe that

$$1 = \sum_{i=1}^{n} P(X_n = i) = \sum_{i=1}^{\lfloor v_k n \rfloor} P(X_n = i) + \sum_{\lfloor v_k n \rfloor}^{n} P(X_n = i) - \underbrace{P(X_n = \lfloor v_k n \rfloor)}_{O(n^{-1/2}) \text{ by Eq. (8)}}.$$

Hence, from what we have just proven, by taking $\ell = 0$ and $t = 0$ in Eq. (6) and Eq. (7), we obtain that

$$\sum_{i=1}^{n} P(X_n = i) \quad \xrightarrow[n\to\infty]{} \quad \frac{E_k \alpha_k\, g(v_k)}{\sqrt{2\pi}} \int_{-\infty}^{\infty} h_0(x)dx.$$

But the left quantity is equal to 1, and the right part can be computed and is equal to $E_k \alpha_k\, g(v_k)\sigma_k$. Hence, $E_k \alpha_k\, g(v_k)\sigma_k = 1$, and after basic simplifications we obtain the following expression for E_k, which is much simpler than Eq. (2):

$$E_k = \frac{1}{\alpha_k\, g(v_k)\sigma_k} = k + \frac{k-1}{v_k}.$$

Note that we only needed to know that E_k exists to obtain the formula above, yielding another proof of Lebensztayn's theorem that does not use Korshunov's complicated expression for E_k.

4 Algorithms for random sampling

In this section we describe random generation algorithms for deterministic and complete accessible automata, which are all variations on the same rejection algorithm[5]. As explained in the introduction, our main algorithm `RandomSampler(`m`,`k`)` (presented in Fig. 5) generates at random structures in $\mathcal{T}_{n,k}$ for $n = \left\lfloor \frac{m}{v_k} \right\rfloor$ and extracts their accessible automata. The algorithm rejects until the accessible automaton is of size m. Recall that the accessible automaton of a structure is obtained by restricting the structure to its set $\{s_1 < \ldots < s_j\}$ of accessible states and by relabeling the state s_i as i for all $i \in [j]$.

```
1 v_k ← 1 + (1/k) W_0(−k e^{−k})
2 n ← ⌊m / v_k⌋
3 repeat
4 |   for (p, a) ∈ [n] × {a_1, ..., a_k} do
5 |   |   δ_T(p, a) ←Uniform([n])
6 |   A ← accessible automaton of T
7 until A has m states
8 return A
```

`RandomSampler(`m`,`k`)`

```
1 v ← e^{−1}
2 repeat
3 |   n ← ⌊m / v⌋
4 |   for (p, a) ∈ [n] × {a_1, ..., a_k} do
5 |   |   δ_T(p, a) ←Uniform([n])
6 |   A ←accessible automaton of T
7 |   v ← max(|A| / n, e^{−1})
8 until A has m states
9 return A
```

`AdaptiveRandomSampler(`m`,`k`)`

Figure 5 Random samplers for deterministic and complete accessible automata. $\delta_T(p, q)$ is the target of the transition starting at p and labeled by a in T.

Let us now analyze the expected complexity of `RandomSampler(`m`,`k`)`. The computation of v_k can be achieved [5] using a truncation of the formula $W_0(x) = \sum_{i=1}^{\infty} \frac{(-i)^{i-1}}{i!} x^i$, which converges exponentially fast for $|x| < \frac{1}{e}$ and hence in particular for $x = -ke^{-k}$. Hence if we keep the m first terms only, we have enough precision for the rest of the algorithm. A breadth-first search algorithm is used to compute the accessible part in time $\Theta(m)$, since k is fixed. The relabeling necessary to obtain the accessible automaton can also be computed in $\Theta(m)$.

Hence the expected complexity is a linear function of m times the expected number of iterations. The expected number of iterations is the expected value of the number of tries to obtain the event $X_n = m$ which is equal to $\frac{1}{P(X_n=m)}$. Using Eq. (10) and Eq. (11), we have that $P(X_n = m)$ is equivalent to $\frac{E_k \alpha_k g(v_k)}{\sqrt{2\pi n}}$. The expected number of iterations is therefore in $\Theta(\sqrt{n})$. This leads to the expected complexity stated in the theorem below.

▶ **Theorem 4.** *For any fixed integer $k \geq 2$, the expected complexity of* `RandomSampler(`m`,`k`)` *is in $\Theta(m^{3/2})$.*

4.1 Approximate Sampling

If we relax the condition of Line 7 in `RandomSampler(`m`,`k`)` to keep A when its number of states is in $[m-\varepsilon\sqrt{m}, m+\varepsilon\sqrt{m}]$, we obtain algorithm `ApproxRandomSampler(`m`,`k`,`ε`)`. Notice that `ApproxRandomSampler(`m`,`k`,`ε`)` outputs automata of different sizes, in $[m - \varepsilon\sqrt{m}, m +$

[5] Some prefer the name "pseudo-algorithm" since it may never halt; but this event has probability 0.

$\varepsilon\sqrt{m}]$. However, if we only consider automata of fixed size $m' \in [m - \varepsilon\sqrt{m}, m + \varepsilon\sqrt{m}]$, `ApproxRandomSampler(`$m$`,`$k$`,`$\varepsilon$`)` is a uniform generator for accessible automata of size m'.

As for `RandomSampler(`m`,`k`)`, the expected complexity is a linear function of m times the average number of iterations, which depends on $P(m - \varepsilon\sqrt{n} \leq X_n \leq m + \varepsilon\sqrt{n})$. For any $\varepsilon > 0$, the average number of iterations of `ApproximateSampling(`m`)` is

$$P(m - \varepsilon\sqrt{n} \leq X_n \leq m + \varepsilon\sqrt{n}) \sim \left(\frac{1}{\sigma_k\sqrt{2\pi}} \int_{-\varepsilon}^{\varepsilon} \exp\left(-\frac{x^2}{2\sigma_k^2}\right) dx\right) = \Theta(1). \tag{16}$$

More precisely, Equation (16) shows that the average number of iterations is in $\Theta(\varepsilon^{-1})$.

▶ **Theorem 5.** *For any fixed integer $k \geq 2$ and real $\varepsilon > 0$, the expected complexity of* `ApproxRandomSampler(`m`,`k`,`ε`)` *is in $\Theta(m)$.*

Remark that the usual approximated range interval is $[m(1-\varepsilon), m(1+\varepsilon)]$: the algorithm we propose is much more precise.

4.2 Avoiding the computation of the constant v_k.

It is possible to avoid the explicit computation of v_k, using the self-adaptive algorithm `AdaptiveRandomSampler(`m`,`k`)` presented in Fig. 5. The idea is that if n is large enough, generating a structure of size n and computing the size n' of its accessible automaton yields an estimation of v_k by $\frac{n'}{n}$, which is likely to be precise. The approximated sampler `AdaptiveApproxRandomSampler(`m`,`k`,`ε`)` is defined similarly.

Though needing more iterations than the first versions[6], these two adaptive algorithms have the same expected complexity as stated in the following theorem.

▶ **Theorem 6.** *For any fixed integer $k \geq 2$ and real $\varepsilon > 0$, the expected complexities of* `AdaptiveRandomSampler(`m`,`k`)` *and* `AdaptiveApproxRandomSampler(`m`,`k`,`ε`)` *are respectively $\Theta(m^{3/2})$ and $\Theta(m)$.*

Note that it could be tempting to replace v_k by a fixed approximation. For instance, one could take 0.8 for v_2. It is easy to show that doing so results in an asymptotically exponential number of rejections. For instance, when taking $v_2 = 0.8$, if we use $f(0.8)^n$ to estimate the proportion of additional rejects, we see that for automata of size 100,000 we do approximatively 7 times as many rejections, but moving to automata of size 1,000,000 this number jumps to approximatively 142,000,000 times as many. This underlines the importance of mathematical analysis not only to study algorithms but also to devise them.

4.3 Sampling random accessible minimal automata

Recently, in [11] it was shown that the probability for an automaton of $\mathcal{A}_{n,k}$ to be minimal tends toward some constant $\lambda_k > 0$ as n tends to infinity.

So if we replace the condition of Line 8 of `RandomSampler(`m`,`k`)` by "$\mathcal{A}$ has m states and is minimal", we obtain a random sampler for accessible and minimal automata. This is strictly equivalent to first use `RandomSampler(`m`,`k`)` and then apply a rejection algorithm to keep minimal automata only; the induced distribution on minimal automata is therefore the uniform distribution.

[6] Simulations for a two-letter alphabet seem to indicate that at most twice as much iterations are required, on average.

If we use Moore's algorithm (which has an average complexity in $O(n \log\log n)$, see Section 5) to test for minimality, we obtain an expected complexity in $\Theta(m^{3/2}+m\log\log m) = \Theta(m^{3/2})$. For the `ApproxRandomSampler(`$m$`,`$k$`,`$\varepsilon$`)`, we obtain an approximated sampler for minimal accessible automata with an expected complexity in $\Theta(m\log\log m)$.

5 Application to analysis of algorithm

The main perspective of this work is to help analyze the average complexities of algorithms that deal with accessible automata using average complexities of this same algorithms on structures.

A common technique for such studies is to isolate sets of inputs with non-typical behaviors, and then prove that the contribution of such sets to the average complexity is negligible, because an input belongs to such a set with a small probability. In our context, it is usually much easier to prove such properties for structures rather than for automata, since they are simpler combinatorial objects. In this section, we briefly describe a general scheme that can be used in these situations: under some general conditions, properties that sufficiently rarely hold for structures still rarely hold for automata.

The idea is the following: let $\mathcal{P}$ be a property of automata (accessible or not) such that if the property holds for the accessible automaton of a structure it also holds for the structure itself. A property such as "being accessible" obviously does not satisfy this requirement but a property such as "having a sink state" does. In the following we explain and illustrate why, if one can afford a $\sqrt{m}$ multiplier, the negligibility of such a property can be transfered from structures to automata.

Let $p_{\mathcal{A}}(m)$ and $p_{\mathcal{T}}(n)$ denote the probabilities that $\mathcal{P}$ holds for a size-m automaton and for a size-n structure, respectively. Then, if A_T denotes the accessible automaton of the structure T,

$$p_{\mathcal{A}}(m) = \frac{|\{T \in \mathcal{T}_n : |A_T| = m \text{ and } A_T \text{ satisfies } \mathcal{P}\}|}{|\{T \in \mathcal{T}_n : |A_T| = m\}|} \leq \frac{|\{T \in \mathcal{T}_n : T \text{ satisfies } \mathcal{P}\}|}{|\{T \in \mathcal{T}_n : |A_T| = m\}|}.$$

The last quantity is equal to $\frac{p_{\mathcal{T}}(n)}{P(X_n=m)}$ by multiplying and dividing the quantity by $|\mathcal{T}_n|$. By taking $n = \left\lfloor \frac{m}{v_k} \right\rfloor$, and using Eq. (8), we obtain that for any such property $\mathcal{P}$,

$$p_{\mathcal{A}}(m) \leq p_{\mathcal{T}}\left(\left\lfloor \frac{m}{v_k} \right\rfloor\right) \cdot O(\sqrt{m}). \tag{17}$$

We now give two examples to illustrate how Eq. (17) can be used. First, we prove that the probability that an automaton has a sink state is asymptotically negligible:

▶ **Lemma 7.** *For the uniform distribution, the probability that an automaton with m states on an alphabet with $k \geq 2$ letters has a sink state is in $O(n^{3/2-k})$.*

Proof. As remarked previously Eq. (17) holds for this property. The probability that a structure with n states has at least one sink state is at most n^{1-k}, since every given state is a sink state with probability n^{-k}. This conclude the proof by taking $n = \lfloor m/v_k \rfloor$. ◀

The same technique can be used to prove a deeper result on Moore's minimization algorithm. David [6] proved that for the uniform distribution on structures with n states on an alphabet with at least two letters, the average complexity of Moore's algorithm is $O(n\log\log n)$.

His result can almost readily be adapted to the uniform distribution on accessible automata, using the method described above. First notice that when applied to a structure T, the complexity of Moore's algorithm is greater than or equal to its complexity for the accessible automaton of T. David's proof relies on showing that the probability that a structure needs more than $\Theta(n \log\log n)$ instructions is small enough to have a negligible contribution to the average complexity. With some care, one can show that his error terms can be handled with the $O(\sqrt{m})$ multiplier of Eq. (17), giving the result.

Acknowledgements: The second author is supported by the ANR project MAGNUM: ANR-2010-BLAN-0204.

References

1. Marco Almeida, Nelma Moreira, and Rogério Reis. Enumeration and generation with a string automata representation. *Theor. Comput. Sci.*, 387:93–102, 2007.
2. Frédérique Bassino, Julien David, and Cyril Nicaud. Enumeration and random generation of possibly incomplete deterministic automata. *Pure Math. and Appl.*, 19:1–16, 2008.
3. Frédérique Bassino and Cyril Nicaud. Enumeration and Random Generation of Accessible Automata. *Theor. Comput. Sci.*, 381:86–104, 2007.
4. Jean-Marc Champarnaud and Thomas Paranthoën. Random generation of DFAs. *Theor. Comput. Sci.*, 330:221–235, 2005.
5. Robert M. Corless, Gaston H. Gonnet, David E. G. Hare, David J. Jeffrey, and Donald E. Knuth. On the Lambert W function. In *Adv. Comp. Math.*, pages 329–359, 1996.
6. Julien David. The Average Complexity of Moore's State Minimization Algorithm is $O(n \log\log n)$. In *Proc. of MFCS '10*, LNCS, pages 318–329, 2010.
7. Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability & Computing*, 13(4-5):577–625, 2004.
8. William Feller. *An Introduction to Probability Theory and Its Applications I.* Wiley, 1968.
9. Philippe Flajolet and Andrew M. Odlyzko. Random mapping statistics. In *Adv. Crypt.*, pages 329–354, 1990.
10. Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics.* Cambridge University Press, 2009.
11. Frédérique Bassino and Julien David and Andrea Sportiello. Asymptotic Enumeration of Minimal Automata. In *Proc. of STACS '12*, LIPIcs, 2012.
12. Irving J. Good. An asymptotic formula for the differences of the powers at zero. *Ann. Math. Statist.*, 32:249–256, 1961.
13. John E. Hopcroft and Jeffrey Ullman. *Introduction to automata theory, languages and computation.* Addison-Wesley, 1980.
14. Aleksey Korshunov. Enumeration of finite automata. *Problemy Kibernetiki*, 34:5–82, 1978.
15. Elcio Lebensztayn. On the asymptotic enumeration of accessible automata. *Discr. Math. & Theor. Comp. Sc.*, 12(3):75–80, 2010.
16. Valery A. Liskovets. The number of initially connected automata. *Cybernetics*, 4:259–262, 1969. english translation of Kibernetika (3) 1969, 16-19.
17. Cyril Nicaud. Comportement en moyenne des automates finis et des langages rationnels. *PhD Thesis, Université Paris VII*, 2000.
18. Albert Nijenhuis and Herbert S. Wilf. *Combinatorial Algorithms.* Academic Press, 1978.

Edge-disjoint Odd Cycles in 4-edge-connected Graphs

Ken-ichi Kawarabayashi[*1] **and Yusuke Kobayashi**[†2]

1 **National Institute of Informatics**
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, Japan
k_keniti@nii.ac.jp

2 **University of Tokyo**
7-3-1, Hongo, Bunkyo-ku, Tokyo, Japan
kobayashi@mist.i.u-tokyo.ac.jp

Abstract

Finding edge-disjoint odd cycles is one of the most important problems in graph theory, graph algorithm and combinatorial optimization. In fact, it is closely related to the well-known max-cut problem. One of the difficulties of this problem is that the Erdős-Pósa property does not hold for odd cycles in general. Motivated by this fact, we prove that for any positive integer k, there exists an integer $f(k)$ satisfying the following: For any 4-edge-connected graph $G = (V, E)$, either G has edge-disjoint k odd cycles or there exists an edge set $F \subseteq E$ with $|F| \leq f(k)$ such that $G - F$ is bipartite. We note that the 4-edge-connectivity is best possible in this statement. Similar approach can be applied to an algorithmic question. Suppose that the input graph G is a 4-edge-connected graph with n vertices. We show that, for any $\varepsilon > 0$, if $k = O((\log \log \log n)^{1/2-\varepsilon})$, then the edge-disjoint k odd cycle packing in G can be solved in polynomial time of n.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases odd-cycles, disjoint paths problem, Erdős-Pósa property, packing algorithm, 4-edge-connectivity

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.206

1 Introduction

Finding edge-disjoint odd cycles is one of the most important problems in combinatorial optimization, graph theory, and graph algorithm. Let us formulate our problem.

The edge-disjoint odd cycle packing
Input. A graph G with n vertices, and an integer k.
Problem. Does G have edge-disjoint k odd cycles?

Let us look at each importance of this problem.

* Research partly supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research, by C & C Foundation, by Kayamori Foundation and by Inoue Research Award for Young Scientists.
† Supported by Grant-in-Aid for Scientific Research and by the Global COE Program "The research and training center for new development in mathematics", MEXT, Japan.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 206–217

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


1.1 Importance in Combinatorial Optimization

In order to consider the edge-disjoint odd cycle packing, it is natural to consider the "fractional" version of the problem. Given a graph G, a *fractional edge-disjoint odd cycle packing* is a function f from $\mathcal{C}$ of odd cycles in G to $[0,1]$ satisfying $\sum_{C:e\in C} f(C) \leq 1$ for each edge e in G. The fractional version of the edge-disjoint odd cycle packing is defined to be maximizing $\sum_{C\in\mathcal{C}} f(C)$ over the fractional odd cycle packings f in G. This allows us to consider the integer programs whose linear program relaxations are duals. One can see that the edge-disjoint odd cycle packing is a "dual" problem of finding a minimum edge cover for the set of all odd cycles, which is one of the most important NP-complete problem, called the maximum cut problem. Fiorini et al. [7] proved that the integrality gap of the edge-disjoint odd cycle packing LP is bounded by a constant for planar graphs. But for general graphs, this is not true. Goemans and Williamson [10] proved that the integrality gap of the dual problem (the odd cycle covering LP) is at most 9/4 for planar graphs.

The edge-disjoint odd cycle packing is known to be NP-hard, even for planar graphs, if k is a part of input, see [7]. We remark that packing *disjoint cycles*, i.e., no parity requirement, has been also studied extensively. It is one of the most fundamental problems in graph theory with applications to several areas (see [2, 18]). For more details in this context, we refer the reader to the book by Schrijver [26].

1.2 Importance in Graph Theory

A family $\mathcal{F}$ of graphs is said to have the *Erdős-Pósa property*, if for every integer k there is an integer $f(k,\mathcal{F})$ such that every graph G contains k edge-disjoint subgraphs each isomorphic to a graph in $\mathcal{F}$ or a set F of at most $f(k,\mathcal{F})$ edges such that $G-F$ has no subgraph isomorphic to a graph in $\mathcal{F}$. The term *Erdős-Pósa property* arose because in [5], Erdős and Pósa proved that the family of cycles (without any parity condition) has this property.

On the other hand, for cycles with odd length, the situation is different. The Erdős-Pósa property does not hold for odd cycles in general. Let us give an example. For a graph G, an *odd cycle cover* is a set of edges $F \subseteq E(G)$ such that $G-F$ is bipartite. An *Escher wall of height* h consists of an elementary wall W of height h and h vertex disjoint paths $P_1, \ldots, P_h$ of length two such that:

(i) Each P_i has both endpoints on W but is otherwise disjoint from W.

(ii) One endpoint of P_i is in the ith brick of the top row of bricks of W, the other is in the $(h+1-i)$th brick of the bottom row of W. Furthermore, both of these vertices are in only one brick of W.

We remark that, as pointed out by Lovász and Schrijver (see [29]), an Escher wall of height h contains neither two edge-disjoint odd cycles nor an odd cycle cover with fewer than h edges. This shows that the Erdős-Pósa property does not hold for odd cycles. However, Reed [21] proved that the Erdős-Pósa property holds for the half integral version of the edge-disjoint odd cycle packing.

1.3 Importance in graph algorithm

The importance of finding edge-disjoint odd cycles comes also from the relation to the edge-disjoint paths problem. In the edge-disjoint paths problem, we are given a graph G and a set of k pairs of vertices (called *terminals*) in G, and we have to decide whether or not G has k edge-disjoint paths connecting given pairs of terminals. This is certainly a central problem in algorithmic graph theory and combinatorial optimization. See surveys [8, 23]. It

has attracted attention in the contexts of transportation networks, VLSI layout and virtual circuit routing in high-speed networks or Internet.

We can see that the k edge-disjoint paths problem can be reduced to finding k edge-disjoint odd cycles as follows. Suppose we have an instance of the edge-disjoint paths problem with a graph $G = (V, E)$ and terminal pairs $(s_1, t_1), \ldots, (s_k, t_k)$. Let G' be the graph obtained from G by subdividing every edge into two edges and by adding an edge connecting s_i and t_i for $i = 1, \ldots, k$. Then finding edge-disjoint k odd cycles in G' is equivalent to finding k edge-disjoint paths in G.

Let us give previous known results on the edge-disjoint paths problem. If k is a part of the input of the problem, then this is known to be NP-complete [6] and it remains NP-complete even if G is constrained to be planar [17]. In fact, even for series-parallel graphs (allowing multiple edges), it remains NP-complete [19]. This is one of the few problems that are known to be NP-complete for series parallel graphs or bounded tree-width graphs. Let us observe that the vertex-disjoint paths problem is solvable for bounded tree-width graphs (and hence for series parallel graphs), see [20].

On the positive side, the seminal work of Robertson and Seymour [24] says that there is a polynomial time algorithm (actually $O(m^3)$ time algorithm, where m is the number of edges of an input graph G) for the edge-disjoint paths problem when the number k of terminals is fixed (the time complexity is improved to $O(n^2)$ in [13] where n is the number of vertices and a shorter correctness proof is given in [16]). Actually, this algorithm is one of the spin-offs of their groundbreaking work on Graph Minor project, spanning 23 papers, and giving several deep and profound results and techniques in discrete mathematics.

Recently, a faster algorithm and a much simpler correctness proof for the edge-disjoint paths problem in 4-edge-connected graphs are given in [12].

▶ **Theorem 1.** *Suppose that the input graph G is 4-edge-connected, which has n vertices. For any $\varepsilon > 0$, if $k = O((\log\log\log n)^{\frac{1}{2}-\varepsilon})$, then the k-edge-disjoint paths problem in G is solvable in polynomial time of n.*

1.4 Main Contributions

Lovász and Schrijver (see [29]) characterized graphs without two edge-disjoint odd cycles. However, their proof heavily depends on the seminal result by Seymour [28] for decomposing regular matroids. No such characterization has been known for k edge-disjoint odd cycles for any fixed k, even $k = 3$. In fact, Lovász and Schrijver considered the problem of finding a structure without many edge-disjoint odd cycles in early 1980's (actually, it seems that Gerards, Seymour, and Thomassen also considered this problem in early 1980's).

As we pointed out, one of the main difficulties is because the Erdős-Pósa property does not hold. The situation is not improved even if we assume a given graph to be 3-edge-connected, as we can easily make the above example 3-edge-connected by adding some parallel edges. On the other hand, if we assume some moderate edge-connectivity, i.e., if we assume 4-edge-connectivity, then the situation dramatically changes. Actually, our result holds also for graphs with no edge-cut of size exactly three, which we call 3-*edge-cut-free graphs*. The following is our main result.

▶ **Theorem 2.** *For any positive integer k, there exists an integer $f(k) = 2^{2^{O(k^2 \log k)}}$ satisfying the following. For any 4-edge-connected graph (or any 3-edge-cut-free graph) $G = (V, E)$, either G has edge-disjoint k odd cycles or there exists an edge set $F \subseteq E$ with $|F| \leq f(k)$ such that $G - F$ is bipartite.*

If we consider "vertex-disjoint" instead of "edge-disjoint", then we need vertex-connectivity $\Theta(k)$ as in [14]. So in the edge-disjoint case, we get a much better result. As we mentioned, the 4-edge-connectivity is best possible.

Similar proof technique for Theorem 2 can be applied to the edge-disjoint odd cycle packing. As we have already seen before, the edge-disjoint k odd cycle packing is a generalization of the k edge-disjoint paths problem. Since the edge-disjoint paths problem in 4-edge-connected graphs is much easier than the problem in general graphs [12], we expect that we can design a simpler algorithm for the edge-disjoint k odd cycle packing under the assumption that the input graph is 4-edge-connected. Here is our second contribution.

▶ **Theorem 3.** *Suppose that the input graph G is a 4-edge-connected graph (or a 3-edge-cut-free graph) with n vertices. For any $\varepsilon > 0$, if $k = O((\log\log\log n)^{1/2-\varepsilon})$, then the edge-disjoint k odd cycle packing in G is solvable in polynomial time of n.*

We have seen that the k edge-disjoint paths problem can be reduced to the edge-disjoint k odd cycle packing by subdividing every edge into two edges and by adding an edge connecting s_i and t_i for $i = 1, \ldots, k$. If the original graph is 4-edge-connected, then the obtained graph is not 4-edge-connected but 3-edge-cut-free. Therefore, Theorem 3 implies Theorem 1 as a corollary.

The characterization by Lovász and Schrijver results in a polynomial time algorithm for testing whether or not a given graph contains two edge-disjoint odd cycles. In general, the following theorem is recently proved.

▶ **Theorem 4** (Kawarabayashi–Reed [15])**.** *For any fixed k, there is a polynomial time algorithm for the edge-disjoint odd cycle packing.*

However, the correctness proof of the algorithm needs the whole graph minor papers, and moreover, a hidden constant is huge.[1] On the other hand, our proof for Theorem 3 is within 5 pages, and the full proof is presented in this paper. In addition, our hidden constant concerning k is not so big and therefore we can handle superconstant concerning k.

It is natural to ask why we do not consider the weaker condition that the minimum degree being at least four, but in fact this weaker restriction would not gain us anything. Consider an instance of the edge-disjoint k odd cycle packing on an arbitrary graph G that may have degree three vertices. Then attach by two edges to each node in G a constant-sized bipartite graph of high minimum degree. This new graph G' has minimum degree high, but the resulting instance of the edge-disjoint k odd cycle packing is clearly equivalent to the original one in G. This example shows that 4-edge-connectivity is necessary. Thus we really need to stick the 4-edge-connectivity in our proof.

2 Preliminaries

In this paper, n and m always mean the numbers of vertices and edges of a given graph, respectively. A pair of subgraphs (A, B) is a *separation* if $G = A \cup B$ and there are no edges in $E(A) \cap E(B)$. The *order* of the separation (A, B) is $|V(A) \cap V(B)|$. We denote a clique (or a complete graph) with t vertices by K_t. A clique minor of order t, denoted by a *K_t-minor*,

[1] To quote David Johnson [11], "for any instance $G = (V, E)$ that one could fit into the known universe, one would easily prefer $|V|^{70}$ to even constant time, if that constant had to be one of Robertson and Seymour's." He estimates one constant in an algorithm for testing for a fixed minor H to be roughly $2 \uparrow 2^{2^{2^{2\uparrow(2\uparrow\Theta(|V(H)|))}}}$, where $2 \uparrow n$ denotes a tower $2^{2^{2^{\cdot^{\cdot^{\cdot}}}}}$ involving n 2's.

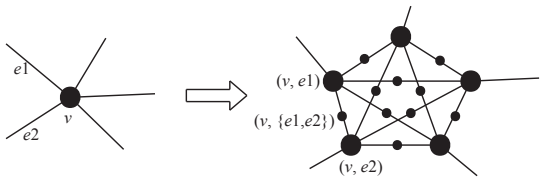


Figure 1 Construction of $L(G)$

can be thought of as t disjoint trees $T_1, \ldots, T_t$ such that there is an edge between T_i and T_j for any i, j with $i \neq j$. Sometimes, one tree T_i is called a *node* of the clique minor. We say that a clique minor K consisting of disjoint trees $T_1, \ldots, T_t$ is *odd*, if for every cycle C in K, $|E(C) \cap (\bigcup_i E(T_i))|$ is even. A *block* of a graph G is a maximal subgraph that is 2-connected (or a single vertex or a K_2).

It is well-known that the edge-disjoint paths problem can be reduced to the vertex-disjoint paths problem by considering the line graph. Similarly, edge-disjoint cycles in a graph correspond to vertex-disjoint cycles in its line graph. However, taking the line graph does not keep the information of parity. Therefore, instead of the line graph, we introduce the *extended line graph*, which is obtained from G by replacing every vertex by a clique whose each edge is subdivided into two edges. More precisely, for a graph $G = (V, E)$, the extended line graph $L(G) = (V^*, E^*)$ of G is defined by

$$
\begin{aligned}
V_1^* &= \{(v, e) \mid v \in V,\ e \in E,\ e \text{ is incident to } v\},\\
V_2^* &= \{(v, \{e_1, e_2\}) \mid v \in V,\ e_1, e_2 \in E,\ e_1 \text{ and } e_2 \text{ are incident to } v\},\\
V^* &= V_1^* \cup V_2^*,\\
E^* &= \{(v, e_1)(v, \{e_1, e_2\}) \mid (v, e_1), (v, \{e_1, e_2\}) \in V^*\}\\
&\quad \cup \{(v_1, e)(v_2, e) \mid e \text{ is an edge connecting } v_1 \text{ and } v_2 \text{ in } G\}.
\end{aligned}
$$

See Figure 1 for the construction of $L(G)$. One can see that G contains edge-disjoint k odd cycles if and only if $L(G)$ contains vertex-disjoint k odd cycles.

We now look at definitions of the tree-width and wall. Let G be a graph, T a tree and let $\mathcal{V} = \{V_t \subseteq V(G) \mid t \in V(T)\}$ be a family of vertex sets $V_t \subseteq V(G)$ indexed by the vertices t of T. The pair $(T, \mathcal{V})$ is called a *tree-decomposition* of G if it satisfies the following three conditions:

- $V(G) = \bigcup_{t \in T} V_t$,
- for every edge $e \in E(G)$ there exists a $t \in T$ such that both ends of e lie in V_t,
- if $t, t', t'' \in V(T)$ and t' lies on the path of T between t and t'', then $V_t \cap V_{t''} \subseteq V_{t'}$.

The *width* of $(T, \mathcal{V})$ is the number $\max\{|V_t| - 1 \mid t \in T\}$ and the *tree-width* $\mathrm{tw}(G)$ of G is the minimum width of any tree-decomposition of G.

We can apply dynamic programming to solve problems on graphs of bounded tree-width, in the same way that we apply it to trees (see e.g. [1]), provided that we are given a bounded width tree decomposition. Bodlaender [3] developed a linear time algorithm.

▶ **Theorem 5.** *For an integer w, there exists a $(w^{O(w)})n^{O(1)}$ time algorithm that, given a graph G, either finds a tree decomposition of G of width w or concludes that the tree-width of G is more than w. Furthermore, if w is fixed, there exists an $O(n)$ time algorithm.*

If the tree-width and k are small, by a standard dynamic programming technique, the edge-disjoint k odd cycle packing can be solved efficiently (see e.g. [1]).

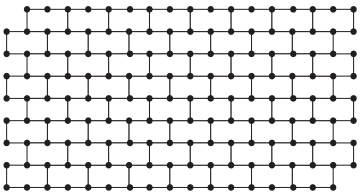


Figure 2 An elementary wall of height 8

▶ **Theorem 6.** *For integers w and k, there exists a $(w^{O(kw)})n^{O(1)}$ time algorithm for the edge-disjoint k odd cycle packing in graphs of tree-width w.*

An elementary wall of height eight is depicted in Figure 1. An *elementary wall* of height h for $h \geq 2$ is similar. It consists of h levels each containing h bricks, where a *brick* is a cycle of length six. A *wall* of height h is obtained from an elementary wall of height h by subdividing some of the edges, i.e. replacing the edges with internally vertex disjoint paths with the same endpoints. The *nails* of a wall are the vertices of degree three within it. Any wall has a unique planar embedding. We define a distance function d_W on the vertices of W so that $d_W(x, y)$ is the minimum number of regions of this embedding that an arc in the plane with endpoints x and y intersects. We define the distance between two subgraphs W_1, W_2 of W by

$$d_W(W_1, W_2) = \min\{d_W(x, y) \mid x \in V(W_1),\ y \in V(W_2)\}.$$

The *perimeter* of a wall W, denoted $\mathrm{per}(W)$, is the boundary of the unique face in this embedding which contains more than six vertices of the original elementary wall. For any wall W in a given graph G, there is a unique component U of $G - \mathrm{per}(W)$ containing $W - \mathrm{per}(W)$. The *compass* of W, denoted $\mathrm{comp}(W)$, is the subgraph of G induced by $V(U) \cup V(\mathrm{per}(W))$. A *subwall* of a wall W is a wall which is a subgraph of W. A subwall of W of height h is *proper* if it consists of h consecutive bricks from each of h consecutive rows of W. For a subgraph H, we say a proper subwall W' is *dividing* in H if H contains W' and the compass of W' in H is disjoint from $(W - W') \cap H$. A wall is *flat* if its compass does not contain two vertex-disjoint paths connecting the diagonally opposite corners. Note that if the compass of W has a planar embedding whose infinite face is bounded by the perimeter of W then W is clearly flat. Seymour [27], Thomassen [30], and others have characterized precisely which walls are flat.

One of the most important results concerning the tree-width is the main result of Graph Minors. V [22] which says the following.

▶ **Theorem 7.** *For any t, there exists a constant $f_1(t)$ such that if G has tree-width at least $f_1(t)$, then G contains a wall W of height t.*

The best known upper bound for $f_1(t)$ is 20^{2t^5}, see [4, 20, 25]. The best known lower bound is $\Theta(t^2 \log t)$, see [25]. Furthermore, such a wall can be found efficiently.

▶ **Theorem 8** ([24, 25])**.** *In a graph G with tree-width at least $f_1(t)$, we can find a wall W of height t in $(f_1(t)^{O(f_1(t))})n^{O(1)}$ time.*

3 Finding a Large Clique Minor

In our algorithm for the edge-disjoint k odd cycle packing, we divide the problem into two cases depending on whether the tree-width of the input graph is large or not. In order to

deal with the case when the tree-width is large, we show the following theorem, which says that we can find a large clique minor in $L(G)$ if the tree-width of G is large.

▶ **Theorem 9.** *For any 4-edge-connected graph (or any 3-edge-cut-free graph) G and for any integer $t \geq 2$, there exists an integer $g(t) = 2^{(2^{O(t^2)})}$ such that one of the following holds:*
1. *G has tree-width at most $g(t)$.*
2. *The extended line graph $L(G)$ contains a clique minor of order t.*

Furthermore, either the tree decomposition of G of width at most $g(t)$ or the K_t-minor in $L(G)$ can be computed in $(g(t)^{O(g(t))})n^{O(1)}$ time.

The objective of this section is to give a proof of this theorem. Since the proof is almost the same as the proof of Theorem 4.1 in [12], we omit it. We note that [12] deals with the line graph instead of the extended line graph.

We now give a remark that a large clique minor plays an important role in the disjoint paths problem. By using the following theorem, we can reduce the disjoint paths problem to an equivalent smaller problem if the input graph has a large clique minor. We will use this theorem also in our proofs of Theorems 2 and 3.

▶ **Theorem 10** (Robertson and Seymour [24, Theorem (5.4)])**.** *Let $s_1, \ldots, s_k, t_1, \ldots, t_k$ be the terminals in a given G. If there is a clique minor of order at least $3k$ in G, and there is no separation (A, B) of order at most $2k-1$ in G such that A contains all the terminals and $B - A$ contains at least one node of the clique minor, then there are vertex-disjoint paths P_i with two ends in s_i, t_i for $i = 1, \ldots, k$.*

4 Erdős-Pósa Property (Proof of Theorem 2)

In this section, we give a proof of Theorem 2. An outline of the proof is described as follows. In Section 4.1, we show that if $L(G)$ has a large clique minor, then G is not a minimum counterexample of Theorem 2. In Section 4.2, we show that if $L(G)$ contains no large clique minor, then G cannot be a counterexample.

4.1 Property of a minimum counterexample

In this subsection, we show that if $L(G)$ has a large clique minor, then G is not a minimum counterexample of Theorem 2. To show this, we use the following theorem given in [9].

▶ **Theorem 11** (Geelen et al. [9, Theorem 13])**.** *There is a constant c such that if G contains a K_t-minor K, where $t \geq \lceil cl\sqrt{\log 12l} \rceil$ for a positive integer l, then one of the following holds.*
1. *G contains an odd K_l-minor.*
2. *There exists a vertex set X with $|X| < 8l$ such that the unique block (i.e., maximal 2-connected subgraph) U of $G - X$ that intersects all the nodes of K disjoint from X is bipartite.*

Furthermore, such an odd K_l-minor or a vertex set X can be found in $O(nm)$ time.

With the aid of this theorem, we show the following property of a minimum counterexample.

▶ **Proposition 12.** *Let k and l be positive integers. Suppose that $G = (V, E)$ is a 4-edge-connected graph (or a 3-edge-cut-free graph) with minimum number of edges such that it does not contain edge-disjoint k odd cycles and $G - F$ is not bipartite for any $F \subseteq E$ with $|F| \leq l$.*

Then, $L(G)$ has no clique minor of order $\max\{\lceil 3ck\sqrt{\log 36k}\rceil, 50k\}$, where c is the constant given in Theorem 11.

Proof. Assume that $L(G)$ has a clique minor K of order $\max\{\lceil 3ck\sqrt{\log 36k}\rceil, 50k\}$. By Theorem 11, we have one of the following.

1. $L(G)$ contains an odd K_{3k}-minor.
2. There exists a vertex set X with $|X| < 24k$ such that the unique block U of $L(G) - X$ that intersects all the nodes of K disjoint from X is bipartite.

When $L(G)$ contains an odd K_{3k}-minor, we can take vertex-disjoint k cycles each passing through three nodes of the clique minor in $L(G)$. Since these cycles are of odd length by the definition of the odd clique minor, we can find vertex-disjoint k odd cycles in $L(G)$. Hence, the corresponding cycles in G are edge-disjoint odd cycles, which contradicts the assumption.

Suppose that there exists a vertex set X with $|X| < 24k$ such that the unique block U of $L(G)-X$ that intersects all the nodes of K disjoint from X is bipartite. Let $U_1, \ldots, U_q$ be the connected component of $L(G) - X - U$ that are not bipartite. Since $L(G)$ does not contain vertex-disjoint k odd cycles, we have $q < k$. By the definition of U, each U_i is adjacent to at most one vertex, say u_i, of U. Therefore, we have a separation (A', B') of $L(G)$ such that $V(A') \cap V(B') = X \cup \{u_1, \ldots, u_q\}$ and $B' - A'$ is a bipartite graph containing $U - \{u_1, \ldots, u_q\}$. We note that $B' - A'$ contains a clique minor of order $50k - |X \cup \{u_1, \ldots, u_q\}| > 25k$. The following claim shows that we can find a separation of small order with some additional conditions.

▶ **Claim 13.** *There exists a separation (A, B) of $L(G)$ with $Y := V(A) \cap V(B)$ such that $|Y| < 25k$, $B - A$ is bipartite, $B - A$ contains a clique minor of order $25k$, we can link up Y by vertex-disjoint paths in any desired way in B, there is no edge with both end vertices in Y, and every vertex in Y is contained in V_1^*, where V_1^* is the vertex set as in the definition of $L(G)$.*

Proof of the claim. Let (A, B) be a separation of $L(G)$ of minimum order such that $V(B) \subseteq V(U) \cup X$, $B - A$ is bipartite, and $B - A$ contains at least one node of the clique minor. Furthermore, we assume that $|B|$ is minimum among such separations. We note that such a separation exists, because (A', B') satisfies these conditions. We show that this separation (A, B) satisfies the conditions in the above claim.

Define $Y = V(A) \cap V(B)$. By the definition of (A, B), it is obvious that $|Y| \leq |X \cup \{u_1, \ldots, u_l\}| < 25k$. Since $B - A$ contains at least one node of the clique minor, at least $50k - |Y| > 25k$ nodes are contained in $B - A$, that is, $B - A$ contains a clique minor K_B of order at least $25k$. By applying Theorem 10 with K_B and the terminal set Y, we can link up Y by vertex-disjoint paths in any desired way in B.

Assume that Y contains a vertex $v^* = (v, \{e_1, e_2\}) \in V_2^*$. Since v^* is adjacent to two vertices, say $v_1^*, v_2^* \in V_1^*$, by removing v^* from Y and adding v_1^* or v_2^*, we can obtain a separation with smaller B, which contradicts the definition of (A, B). Thus, we have $Y \subseteq V_1^*$. Furthermore, there exists no edge with both end vertices in Y, because we can remove one end vertex from Y if such an edge exists. ◀

Let (A, B) and Y be as in this claim, and we denote $Y = \{(v_i, e_i) \mid i = 1, 2, \ldots, |Y|\}$ because $Y \subseteq V_1^*$. Let $F_Y = \{e_1, \ldots, e_{|Y|}\}$ be the edge set in G that corresponds to Y. Then, one component H of $G - F_Y$ corresponds to $B - A$. More precisely, if $(v, e) \in V_1^*$ is contained in $B - A$, then the corresponding edge e is contained in H. Now we observe the following by the properties of $B - A$ and the definition of $L(G)$.

1. Since $B - A$ is bipartite, H is also bipartite.

2. Since we can link up Y by vertex-disjoint paths in any desired way in B, we can connect F_Y in H by edge-disjoint paths in any desired way. We note that every path in B connecting a fixed pair of vertices in Y has the same parity, and the same thing holds for paths connecting F_Y in H.
3. Since $B - A$ contains at least $25k + 1$ nodes of the clique minor, H contains at least $25k + 1$ edges.

Since H is bipartite by the first property, $V(H)$ is partitioned into two color classes $V_1(H)$ and $V_2(H)$. Now we contract $V_i(H)$ to a single vertex v_i for $i = 1, 2$ in G, and we remove some edges between v_1 and v_2 so that $25k$ edges of $E(H)$ remain between them. Let $\hat{G} = (\hat{V}, \hat{E})$ be the obtained graph. We note that F_Y might contain an edge between two vertices in $V_1(H)$ (or $V_2(H)$), because B is not necessarily bipartite. In such a case, $\hat{G}$ contains a self-loop. We can see that this reduction does not affect the existence of edge-disjoint k odd cycles by the second property. On the other hand, if $\hat{G} - \hat{F}$ is bipartite for some $\hat{F} \subseteq \hat{E}$ with $|\hat{F}| \leq l$, then we can make G bipartite by removing the edge set that corresponds to $\hat{F}$, which contradicts the assumption. Note that $\hat{F}$ does not contain an edge connecting v_1 and v_2, because the number of edges between v_1 and v_2 is bigger than $|F_Y|$. Thus, $\hat{G} - \hat{F}$ is not bipartite for any $\hat{F} \subseteq \hat{E}$ with $|\hat{F}| \leq l$. Since the number of edges decreases by the third property, this contradicts the minimality of G. ◀

4.2 When $L(G)$ has no large clique minor

In this subsection, we consider the case when $L(G)$ has no clique minor of order $t = \max\{\lceil 3ck\sqrt{\log 36k} \rceil, 50k\}$. By Theorem 9, the tree-width of G is bounded by some constant $g(t)$. Since each vertex of G has degree at most t, one vertex in G is replaced by at most $t + \binom{t}{2} < t^2$ vertices in $L(G)$. Thus, the tree-width of $L(G)$ is smaller than a constant $t^2 g(t)$. Now we show the following proposition.

▶ **Proposition 14.** *Let k be a positive integer. Let t and $g(t)$ be positive integers as above, and suppose that there exists an integer $f(i)$ satisfying the condition in Theorem 2 for $i = 1, 2, \ldots, k-1$. Suppose that G is a 4-edge-connected graph (or a 3-edge-cut-free graph) not containing edge-disjoint k odd cycles such that $L(G)$ has no clique minor of order t, and $F \subseteq E$ is a minimum edge set such that $G - F$ is bipartite. Then, we have $|F| \leq \max\{4f(k-1), 3t^2 g(t)\}$.*

Proof. Assume that $|F| > \max\{4f(k-1), 3t^2 g(t)\}$. First, we show that F is "highly-connected" in some sense. For two edges e_1, e_2, we say that a path P *connects* e_1 and e_2 if the first and last edges of P are e_1 and e_2. A path connects two edge sets F_1 and F_2 if it connects edges in F_1 and F_2. We show the following claim.

▶ **Claim 15.** *For any sets $F', F'' \subseteq F$ with $|F'| = |F''| \leq |F|/2$, there exist $|F'|$ edge-disjoint paths each connecting F' and F''.*

Proof of the claim. Let $F', F'' \subseteq F$ be sets with $|F'| = |F''| \leq |F|/2$. To derive a contradiction, assume that G does not contain $|F'|$ edge-disjoint paths connecting them. By Menger's theorem, there exists an edge set $C \subseteq E$ such that $|C| \leq |F'| - 1$ and $G - C$ contains no path connecting F' and F''. That is, there exists a partition (G_1, G_2) of $G - C$ such that $V(G_1 \cap G_2) = \emptyset$, $F' \subseteq E(G_1) \cup C$, and $F'' \subseteq E(G_2) \cup C$.

If both G_1 and G_2 contain an odd cycle, then each has at most $k-2$ edge-disjoint odd cycles. By induction hypothesis, for $i = 1, 2$, G_i has an edge set F_i with $|F_i| \leq f(k-1)$ such that

$G_i - F_i$ is bipartite. Then, $G-(F_1 \cup F_2 \cup C)$ is bipartite and $|F_1 \cup F_2 \cup C| < 2f(k-1)+\frac{|F|}{2} \leq |F|$, which contradicts the minimality of F.

Suppose that G_1 contains no odd cycle. Then, $G - ((F \setminus F') \cup C)$ is bipartite and $|(F \setminus F') \cup C| < |F|$, which contradicts the minimality of F. The same argument can be applied when G_2 contains no odd cycle. ◀

We choose one end vertex v_e arbitrary for each $e \in F$, and define a vertex set $V_F = \{(v_e, e) \mid e \in F\}$ of $L(G)$. Then, V_F is $|F|/2$-connected in $L(G)$ by the above claim, where we say that a vertex set X is *κ-connected* if $|X| \geq \kappa$ and for all subsets $X_1, X_2 \subseteq X$ with $|X_1| = |X_2| \leq \kappa$ there are $|X_1|$ vertex-disjoint paths connecting X_1 and X_2. In particular, since $|F| > \max\{4f(k-1), 3t^2g(t)\}$, V_F is a $\frac{3}{2}t^2g(t)$-connected set of size $3t^2g(t)$. Now we use the following lemma.

▶ **Lemma 16** (Diestel et al. [4, Proposition 3]). *Let G be a graph and κ be a positive integer. If G contains a $(\kappa+1)$-connected set of size at least 3κ, then G has tree-width at least κ.*

Since $L(G)$ has a $\frac{3}{2}t^2g(t)$-connected set V_F of size $3t^2g(t)$, $L(G)$ has tree-width at least $t^2g(t)$ by Lemma 16, which is a contradiction. ◀

By Propositions 12 and 14, $f(k)$ is bounded by $3t^2g(t) = 2^{2^{O(k^2 \log k)}}$, which completes the proof of Theorem 2.

5 Packing Algorithm (Proof of Theorem 3)

In this section, we give an algorithm for the edge-disjoint k odd cycle packing in 4-edge-connected graphs (or 3-edge-cut-free graphs), and prove Theorem 3.

Since the case with small tree-width is easy by Theorem 6, it suffices to deal with the case when the extended line graph $L(G)$ has a large clique minor by Theorem 9. For this case, we give a procedure that reduces the original instance to a smaller instance.

▶ **Proposition 17.** *Let G be a 4-edge-connected graph (or a 3-edge-cut-free graph) and k be a positive integer. If a clique minor of order at least $\max\{\lceil 3ck\sqrt{\log 36k}\rceil, 50k\}$ is given in $L(G)$, where c is the constant given in Theorem 11, then we can reduce an instance of the edge-disjoint k odd cycle packing in G to an equivalent smaller instance in polynomial time.*

Proof. We use a similar argument to the proof of Proposition 12. Suppose that the extended line graph of G contains a clique minor K of order at least $\max\{\lceil 3ck\sqrt{\log 36k}\rceil, 50k\}$. Then, we have one of the following by Theorem 11.

1. $L(G)$ contains an odd K_{3k}-minor.
2. There exists a vertex set X with $|X| < 24k$ such that the unique block U of $L(G) - X$ that intersects all the nodes of K disjoint from X is bipartite.

When $L(G)$ contains an odd K_{3k}-minor, G contains edge-disjoint k odd cycles, which means that we can easily find edge-disjoint k odd cycles in G. (In other words, we can reduce the original instance to a trivial "YES" instance.)

Suppose that there exists a vertex set X with the above conditions. Let $A, B, Y, F_Y, H, V_1(H)$, and $V_2(H)$ be as in the proof of Proposition 12. Construct a smaller graph by contracting $V_i(H)$ to a single vertex v_i for $i = 1, 2$ and by removing some edges between v_1 and v_2, and let $\hat{G} = (\hat{V}, \hat{E})$ be the obtained graph. We have already seen in the proof of Proposition 12 that this reduction does not affect the existence of edge-disjoint k odd cycles. Since the obtained graph is smaller than the original one, the obtained instance is a desired one. ◀

Now we give a proof of Theorem 3.

Proof of Theorem 3. First, we apply Theorem 9 where $t = \max\{\lceil 3ck\sqrt{\log 36k}\rceil, 50k\}$ and c is the constant given in Theorem 11. Then, either the input graph G has tree-width at most $g(t)$, or $L(G)$ contains a clique minor of order t by Theorem 9. In the first case, we can solve the edge-disjoint k odd cycle packing in G in $(g(t)^{O(kg(t))})n^{O(1)}$ time by Theorem 6. In the second case, we apply Proposition 17 to obtain a smaller instance, and recurse the algorithm. We note that the running time $(g(t)^{O(kg(t))})n^{O(1)}$ is bounded by a polynomial of n if $k = O((\log\log\log n)^{1/2-\varepsilon})$. This shows that we can solve the problem in polynomial time. ◀

Finally, we give a remark on the time complexity for the case when k is a fixed constant. In this case, the most time consuming part is to execute Theorem 11 repeatedly. Since $L(G)$ might have $\Omega(n^3)$ vertices and edges when G has vertices of high degree, if we apply a naive reduction algorithm, then the time complexity of the reduction step becomes $O(n^6)$, and so the total running time is $O(n^6 m)$. If we find an edge set F_Y of G directly (i.e., without constructing $L(G)$), then the running time will be greatly improved, but we will not discuss this issue in this paper.

References

1 S. Arnborg and A. Proskurowski, Linear time algorithms for NP-hard problems restricted to partial k-trees, *Discrete Appl. Math.*, **23** (1989), 11–24.

2 P. Balister, Packing digraphs with directed closed trails, *Combin. Probab. Comput.*, **12** (2003), 1–15.

3 H.L. Bodlaender, A linear-time algorithm for finding tree-decomposition of small treewidth, *SIAM J. Comput.*, **25** (1996), 1305–1317.

4 R. Diestel, K.Y. Gorbunov, T.R. Jensen and C. Thomassen, Highly connected sets and the excluded grid theorem, *J. Combin. Theory Ser. B*, **75** (1999), 61–73.

5 P. Erdős and L. Pósa, On the independent circuits contained in a graph, *Canadian Journal of Mathematics*, **17** (1965), 347–352.

6 S. Even, A. Itai and A. Shamir, On the complexity of timetable and multicommodity flow problems, *SIAM J. Comput.*, **5** (1976), 691–703.

7 S. Fiorini, N. Hardy, B. Reed, and A. Vetta, Approximate min-max relations for odd cycles in planar graphs, *Mathematical Programming*, **110** (2007), 71–91.

8 A. Frank, Packing paths, cuts and circuits – a survey, in *Paths, Flows and VLSI-Layout*, B. Korte, L. Lovász, H.J. Promel and A. Schrijver (Eds.), Springer-Verlag, Berlin, 1990, 49–100.

9 J. Geelen, B. Gerards, B. Reed, P. Seymour, and A. Vetta, On the odd-minor variant of Hadwiger's conjecture, *J. Combin. Theory Ser. B*, **99** (2009), 20–29.

10 M.X. Goemans and D.P. Williamson, Primal-dual approximation algorithms for feedback problems in planar graphs, *Combinatorica*, **18** (1998), 37–59.

11 D. Johnson, The many faces of polynomial time, in The NP-completeness column: An ongoing guide, *J. Algorithms*, **8** (1987), 285–303.

12 K. Kawarabayashi and Y. Kobayashi, The edge disjoint paths problem in Eulerian graphs and 4-edge-connected graphs, *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010, 45–353.

13 K. Kawarabayashi, Y. Kobayashi, and B. Reed, The disjoint paths problem in quadratic time, *J. Combin. Theory Ser. B*, to appear.

14 K. Kawarabayashi and B. Reed, Highly parity linked graphs, *Combinatorica*, **29** (2009), 215–225.

15 K. Kawarabayashi and B. Reed, Odd cycle packing, *Proc. 42nd ACM Symposium on Theory of Computing (STOC)*, 2010, 695–704.

16 K. Kawarabayashi and P. Wollan, A shorter proof of the graph minors algorithm – the unique linkage theorem, *Proc. 42nd ACM Symposium on Theory of Computing (STOC)*, 2010, 687–694. A full version of this paper is available at `http://research.nii.ac.jp/~k_keniti/uniquelink.pdf`.

17 M.R. Kramer and J. van Leeuwen, The complexity of wire-routing and finding minimum area layouts for arbitrary VLSI circuits, *Adv. Comput. Res.*, **2** (1984), 129–146.

18 M. Krivekevich, Z. Nutov, M.R. Salavatipour, J. Verstaete, and R. Yuster, Approximating algorithms and hardness results for cycle packing problems, *ACM transaction on Algorithms*, **3** (2007), article 48.

19 T. Nishizeki, J. Vygen, and X. Zhou, The edge-disjoint paths problem is NP-complete for series-parallel graphs, *Discrete Appl. Math.*, **115** (2001), 177–186.

20 B. Reed, Tree width and tangles: a new connectivity measure and some applications, in *Surveys in Combinatorics*, London Math. Soc. Lecture Note Ser. **241**, Cambridge Univ. Press, Cambridge, 1997, 87–162.

21 B. Reed, Mangoes and blueberries, *Combinatorica*, **19** (1999), 267–296.

22 N. Robertson and P.D. Seymour, Graph minors. V. Excluding a planar graph, *J. Combin. Theory Ser. B*, **41** (1986), 92–114.

23 N. Robertson and P.D. Seymour, An outline of a disjoint paths algorithm, in *Paths, Flows, and VLSI-Layout*, B. Korte, L. Lovász, H.J. Prömel, and A. Schrijver (Eds.), Springer-Verlag, Berlin, 1990, 267–292.

24 N. Robertson and P.D. Seymour, Graph minors. XIII. The disjoint paths problem, *J. Combin. Theory Ser. B*, **63** (1995), 65–110.

25 N. Robertson, P.D. Seymour, and R. Thomas, Quickly excluding a planar graph, *J. Combin. Theory Ser. B*, **62** (1994), 323–348.

26 A. Schrijver, *Combinatorial Optimization: Polyhedra and Efficiency*, Springer-Verlag, 2003.

27 P.D. Seymour, Disjoint paths in graphs, *Discrete Math.*, **29** (1980), 293–309.

28 P.D. Seymour, Decomposition of regular matroids, *J. Combin. Theory, Ser. B*, **28** (1980), 305–359.

29 P. Seymour, Matroid minors, in *Handbook of Combinatorics*, R.L. Graham, M. Grötschel, and L. Lóvasz (Eds.), North-Holland, Amsterdam, 1985, 419–431.

30 C. Thomassen, 2-linked graph, *European Journal of Combinatorics*, **1** (1980), 371–378.

Efficient algorithms for highly compressed data: The Word Problem in Higman's group is in P

Volker Diekert[1], Jürn Laun[1], and Alexander Ushakov[2]

1 **FMI, Universität Stuttgart**
Universitätsstr. 38, 70569 Stuttgart, Germany
{diekert,laun}@fmi.uni-stuttgart.de

2 **Department of Mathematics, Stevens Institute of Technology**
Hoboken, NJ 07030, USA
sasha.ushakov@gmail.com

Abstract

Power circuits are data structures which support efficient algorithms for highly compressed integers. Using this new data structure it has been shown recently by Myasnikov, Ushakov and Won that the Word Problem of the one-relator Baumslag group is is decidable in polynomial time. Before that the best known upper bound was non-elementary. In the present paper we provide new results for power circuits and we give new applications in algorithmic group theory: 1. We define a modified reduction procedure on power circuits which runs in quadratic time thereby improving the known cubic time complexity. 2. We improve the complexity of the Word Problem for the Baumslag group to cubic time thereby providing the first practical algorithm for that problem. 3. The Word Problem of Higman's group is decidable in polynomial time. It is due to the last result that we were forced to advance the theory of power circuits.

Thanks. Part of this work was done when the first two authors visited the Stevens Institute of Technology in September 2010 and March 2011. The support and the hospitality of the Stevens Institute is greatly acknowledged. The work of the third author was partially supported by NSF grant DMS-0914773.

1998 ACM Subject Classification F.2.2 Computations on discrete structures, G.2.2 Graph Algorithms

Keywords and phrases Algorithmic group theory, Data structures, Compression, Word Problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.218

1 Introduction

Power circuits have been introduced in [18] as a data structure for integers which supports $+$, $-$, $\leq$, and $(x, y) \mapsto 2^x y$. Thus, by iteration it is possible to represent, by very small circuits, huge values (involving the tower function). Efficient algorithms for power circuits yield efficient algorithms for arithmetic with integers in highly compressed form. This idea of *efficient algorithms for highly compressed data* is the main underlying theme of the present paper. In this sense our paper is simultaneously about compression, data structures and about algorithmic group theory.

In 1910 Max Dehn [5] formulated fundamental algorithmic problems for groups. The most prominent one is the *Word Problem:* "Given a finite presentation of some fixed group G, decide whether an input word w represents the trivial element 1_G in G." It took until the 1950's that Novikov and Boone constructed (independently) finitely presented groups with an undecidable Word Problem [21, 3]. There are also finitely presented groups with a

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 218–229

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

decidable Word Problem but with arbitrarily high complexity [25, Theorem 1.3]. In these examples the difficult instances are extremely sparse and, inherently due to the constructions, these groups never appear in any natural setting.

A finitely presented group has a decidable word problem if and only if there is a recursive upper bound on its Dehn function. Although the Dehn function gives a lot more of information about the group (e.g., if it is linear, then the group is hyperbolic and the Word Problem is linear), the Dehn function is not necessarily a good indicator for the complexity of the Word Problem [16, 23]. However, for "natural examples" the connection between the Dehn function and the complexity of the Word Problem was believed to be rather tight.

Such a natural example was the Baumslag group $G_{(1,2)}$ (sometimes called Baumslag-Gersten group). It is a non-cyclic one-relator group all of whose finite factor groups are cyclic [1]. Being a one-relator group the word problem is decidable. However, the only known general way to solve the Word Problem in one-relator groups is by a so-called Magnus break-down procedure [17, 15] which computes normal forms. It was developed in the 1930s and there has been no progress ever since. Its time-complexity on $G_{(1,2)}$ is non-elementary, since it cannot be bounded by any tower of exponents. Actually, Gersten showed that the Dehn function of $G_{(1,2)}$ is non-elementary [9], see also [24]. Therefore (until recently) $G_{(1,2)}$ was the simplest candidate for a group with a non-polynomial Word Problem in the worst case. But then it turned out that its Word Problem is in P: Using the ability of power circuits to compress huge numbers, Myasnikov, Ushakov and Won showed that the Word Problem of the Baumslag group is solvable in polynomial time [19].

It should be noted that the question of algorithmic hardness of the Word Problem in one-relator groups is still wide open, but some researchers conjecture that it is polynomial (even quadratic, see [2]), based on observations on generic-case complexity [11].

The contributions of the present paper are as follows: In a first part, we give new efficient manipulations of the data structure of *power circuits*. We improves the complexity of the reduction algorithm of power circuits from cubic to quadratic time. With the help of this improved reduction algorithm (and some other ideas) we can, as our second result, reduce the complexity of the Word Problem in $G_{(1,2)}$ significantly from $\mathcal{O}(n^7)$ in [19] down to $\mathcal{O}(n^3)$. This cubic algorithm is the first practical algorithm which works for that problem on all reasonably short instances. The algorithm has been implemented and tested. It is available in the CRAG library [20].

Another new application of power circuits shows that the Word Problem in Higman's group H_4 is decidable in polynomial time. This is our third and main result. Higman's group H_4 is a very interesting group with 4 generators and 4 simple defining relations. Higman [10] constructed H_4 in 1951 as the first example of a finitely presented infinite group where all finite quotient groups are trivial. This leads immediately to an infinite simple group which is finitely generated; and no such group was known before Higman's construction. The group H_4 is constructed by amalgamation (see Section 5 or [27]), which yields decidability of the Word Problem, but the procedure computes normal forms and the length of normal forms can be a tower function in the input length. Thus, Higman's group was another natural, but rather complicated candidate for a finitely presented group with an extremely hard Word Problem. Our paper eliminates H_4 as a candidate: We show that the Word Problem of H_4 is in $\mathcal{O}(n^6)$. Actually, the algorithm for H_4 is more complicated than for the Baumslag group $G_{(1,2)}$.

We obtain this result by new techniques for efficient manipulations of multiple markings in a single power circuit and their ability for huge compression rates. Compression techniques have been applied elsewhere for solving word problems, [12, 13, 26]. But in these papers

the authors use straight-line programs whose compression rates are far too small (at best exponential) to cope with Baumslag or Higman groups.

Due to lack of space in this conference version of the paper, we present a slightly less efficient, yet much less technical version of the reduction procedure. In formal statements we use the "soft-$\mathcal{O}$ notation". Full proofs for the complexity bounds without the poly-logarithmic factors as stated e.g. in the abstract can be found online [7].

Algorithms and problems are classified by their *time complexity* on a random-access machine (RAM).

The *tower function* $\tau : \mathbb{N} \to \mathbb{N}$ is defined as usual: $\tau(0) = 1$ and $\tau(i+1) = 2^{\tau(i)}$ for $i \geq 0$. For instance $\tau(4) = 2^{2^{2^{2^1}}} = 2^{16}$ and $\tau(6)$ written in binary cannot be stored in the memory of any conceivable real-world computer. We use standard notation and facts from group theory as found in the classical text book [15].

2 Power circuits

This section is based on [18], but with improved time complexities. In addition, we provide new material such as our treatment of multiple markings which makes the data structure more versatile. This is used for our results on Higman's group. Let Γ be a set and δ be a mapping $\delta : \Gamma \times \Gamma \to \{-1, 0, +1\}$. This defines a directed graph (Γ, Δ), where Γ is the set of vertices and the set of directed arcs (or edges) is $\Delta = \sigma\delta = \{(P, Q) \in \Gamma \times \Gamma \mid \delta(P, Q) \neq 0\}$ (the *support* of the mapping δ). Note that the sign of $\delta(P, Q)$ is to be read as the edge's label and has nothing to do with its orientation. Throughout we require that (Γ, Δ) is a *dag* (*directed acyclic graph*). In particular, $\delta(P, P) = 0$ for all vertices P.

A *marking* is a mapping $M : \Gamma \to \{-1, 0, +1\}$. We can also think of a marking as a subset of Γ where each element in M has a sign (+ or −). (Thus, we also speak about a *signed subset*.) Each node $P \in \Gamma$ is associated with a marking, which is called its Λ-marking or successor marking Λ_P, consisting of the target nodes of outgoing arcs from P:

$$\Lambda_P : \Gamma \to \{-1, 0, +1\}\,,\ Q \mapsto \delta(P, Q)$$

Thus, the marking Λ_P is the signed subset which corresponds to the targets of outgoing arcs from P. We define the *evaluation* $\varepsilon(P)$ of a node ($\varepsilon(M)$ of a marking resp.) by imposing:

$$\varepsilon(P) = 2^{\varepsilon(\Lambda_P)} \quad \text{for a node } P, \qquad \varepsilon(M) = \sum_{P \in \Gamma} M(P)\varepsilon(P) \quad \text{for a marking } M.$$

Leaves evaluate to 1. The values $\varepsilon(P)$ and $\varepsilon(M)$ can be computed bottom-up in the dag, making $\varepsilon(P)$ and $\varepsilon(M)$ well-defined real numbers. The evaluation of a node P is positive.

▶ **Definition 1.** A *power circuit* is a pair $\Pi = (\Gamma, \delta)$ with $\delta : \Gamma \times \Gamma \to \{-1, 0, +1\}$ such that (Γ, Δ) is a dag as above with the additional property that $\varepsilon(M) \in \mathbb{Z}$ for all markings M.

We will see in Corollary 8 that it is possible to check in quasi-quadratic time whether a dag (Γ, Δ) is a power circuit. (One checks $\varepsilon(\Lambda_P) \geq 0$ for all nodes P.)

▶ **Example 2.** We can represent every integer in the range $[-n, n]$ as the evaluation of some marking in a power circuit with node set $\{P_0, \ldots, P_\ell\}$ such that $\varepsilon(P_i) = 2^i$ for $0 \leq i \leq \ell$ and $\ell = \lfloor \log_2 n \rfloor$. Thus, we can convert the binary notation of an integer n into a power circuit with $\mathcal{O}(\log |n|)$ vertices and $\mathcal{O}((\log |n|) \log\log |n|)$ arcs.

▶ **Example 3.** A power circuit can realize tower functions, since a chain of $n+1$ nodes allows us to represent $\tau(n)$ as the evaluation of the last node.

We denote the empty marking (the constant zero mapping) by 0 and for any marking M there is an obvious definition of $-M$ having $\varepsilon(-M) = -\varepsilon(M)$. The insertion of a new node $\textsc{Clone}(P)$ without incoming arcs and with $\Lambda_{\textsc{Clone}(P)} = \Lambda_P$ is called *cloning of a node* P. The notion is extended to markings, where $\textsc{Clone}(M)$ is obtained by cloning all nodes in $\sigma(M)$ and defining $\textsc{Clone}(M)(\textsc{Clone}(P)) = M(P)$ for $P \in \sigma(M)$ and $\textsc{Clone}(M)(P) = 0$ otherwise. We say that M is a *source*, if no node in $\sigma(M)$ has any incoming arcs. Note that $\textsc{Clone}(M)$ is always a source.

If M and K are markings, then $M + K$ given by $(M + K)(P) = M(P) + K(P)$ is a mapping where -2 and 2 may appear as images. For every P with $M(P) + K(P) = \pm 2$, let $P' = \textsc{Clone}(P)$ and redefine $M + K$ by putting $(M + K)(P) = (M + K)(P') = \pm 1$. In this way we can realize addition (and subtraction) in a power circuit by cloning at most $|\sigma(M) \cap \sigma(K)|$ nodes. Note that any other marking in the power circuit remains unaffected by this operation.

Next, consider markings U and X with $\varepsilon(U) = u$ and $\varepsilon(X) = x$ such that $u2^x \in \mathbb{Z}$ (e.g. due to $x \geq 0$). We obtain a marking V with $\varepsilon(V) = u2^x$ and $|\sigma(V)| = |\sigma(U)|$ as follows. First, let $V = \textsc{Clone}(U)$ and $X' = \textsc{Clone}(X)$. Next, introduce additional arcs from every $P' \in \sigma(V)$ to every $Q' \in \sigma(X')$ with signs given by $\delta(P', Q') = X'(Q')$. Note that the cloning of X avoids double arcs from V to X. The cloning of U is not necessary, if U happens to be a source.

We now introduce the reduction of a power circuit which allows us to compare markings.

▶ **Definition 4.** A *reduced power circuit* consists of

i) a power circuit $\Pi = (\Gamma, \delta)$ in which no two different nodes evaluate to the same number,
ii) an ordered list $[P_1, \ldots, P_n]$ of the nodes Γ such that $\varepsilon(P_i) < \varepsilon(P_{i+1})$ for all $1 \leq i < n$,
iii) a bit vector $[b(1), \ldots, b(n-1)]$ where $b(i) = 1$ if and only if $2\varepsilon(P_i) = \varepsilon(P_{i+1})$.

▶ **Proposition 5** ([18]). There is an $\mathcal{O}(|\Gamma|)$ time algorithm which on input a reduced power circuit and two markings K and M compares $\varepsilon(K)$ and $\varepsilon(M)$. It outputs whether the two values are equal and if not, which one of them is larger. In the latter case it also tells whether $|\varepsilon(K) - \varepsilon(M)|$ is exactly 1 or ≥ 2. (This is essentially an argument about binary sums $\sum_{i \geq 0} a_i \cdot 2^i$ with $a_i \in \{-1, 0, +1\}$.) ◀

Algorithm 1 ExtendReduction

Input: A dag $\Pi = (\Gamma \dot\cup U, \delta)$ with no arcs pointing from Γ to U, such that $(\Gamma, \delta|_{\Gamma\times\Gamma})$ is a reduced power circuit and a list $\mathcal{M}$ of markings of Π.
Output: The output of the procedure is "no", if Π is not a power circuit (because $\varepsilon(P) \notin \mathbb{Z}$ for some node P). Else, the output is a reduced power circuit $\Pi' = (\Gamma', \delta')$ and a list $\mathcal{M}'$ of markings of Π' where:

i) $\Gamma \subseteq \Gamma'$ and $\delta|_{\Gamma\times\Gamma} = \delta'|_{\Gamma\times\Gamma}$
ii) $|\Gamma'| \leq |\Gamma| + 3\,|U|$
iii) For all $Q \in U$ there is a node $Q' \in \Gamma'$ with $\varepsilon(Q) = \varepsilon(Q')$.
iv) For every marking $M \in \mathcal{M}$ there is a corresponding marking $M' \in \mathcal{M}'$ with $\varepsilon(M') = \varepsilon(M)$ and $|\sigma(M')| \leq |\sigma(M)|$.

```
1 compute a topological order [Q_1, ..., Q_|U|] of U, i.e., an ordering of the nodes such
      that for i ≤ j there are no arcs from Q_i to Q_j;
2 for i = 1, ..., |U| do
3     U := U \ {Q_i};
4     let [P_1, P_2, ...] be the ordered list of the current node set Γ;
```

```
5   using binary search, find the minimal j such that
        ε(Q_i) ≤ ε(P_j); /* check ε(Λ_{Q_i}) ≤ ε(Λ_{P_j}) */
6   if ε(Λ_{Q_i}) < 0 then return no fi;
7   if ε(Q_i) < ε(P_j) then /* check ε(Λ_{Q_i}) < ε(Λ_{P_j}) */
8       Γ := Γ ∪ {Q_i};
9       insert {Q_i} into Γ's sorted list of nodes between P_j and P_{j+1};
10      set the bit vector for Q_i according to whether ε(Λ_{Q_i}) + 1 = ε(Λ_{P_j})
11  else /* ε(Q_i) = ε(P_j) */
12      find the maximum n such that b(1) = ... = b(n − 1) = 1;
13      as in example 2, create a new node B with ε(B) = 2^{n+1} and adjust the
            ordered list of Γ and the bit vector accordingly;
14      using the ordered list of Γ and the bit vector, find the maximal number k
            such that there is a chain of nodes P_j = R_0, R_1, ..., R_{k−1} with
            ε(R_ℓ) = ε(P_j) · 2^ℓ (for 0 ≤ ℓ < k);
15      R_k := CLONE(R_{k−1});
16      find the maximal ℓ such that Λ_{R_k}(P_1) = ... = Λ_{R_k}(P_{ℓ−1}) = +1;
17      remove P_1, ..., P_{ℓ−1} from Λ_{R_k} and set Λ_{R_k}(P_ℓ) := +1 instead;
18      insert R_k into Γ's ordered list between P_{j+k−1} and P_{j+k} and set the bit
            vector for R_k according to whether ε(Λ_{R_k}) + 1 = ε(Λ_{P_{j+k}})
19      foreach M ∈ M ∪ {Λ_Q : Q ∈ U} with M(Q_i) =: s ≠ 0 do
20          remove Q_i from M;
21          ℓ := 0; while M(R_ℓ) = s do remove R_ℓ from M od;
22          let M(R_ℓ) := M(R_ℓ) + s
23      od fi od
```

▶ **Theorem 6.** *The procedure* ExtendReduction *given in Algorithm 1 is correct and runs in time* $\widetilde{\mathcal{O}}((|\Gamma| + |U|) \cdot |U| + |\sigma(M_1)| + \ldots + |\sigma(M_m)|)$*, where* $M_1, \ldots, M_m$ *are the markings in* $\mathcal{M}$ *whose support contains nodes in* U*.*

Proof. In the outer loop, the nodes are moved from U to Γ one by one. The topological order ensures that arcs originating from the currently processed node Q_i all end in Γ. The binary search used to determine the right place of U_i inside Γ takes $\log(|\Gamma|+|U|)$ comparisons (remember that $|\Gamma|$ grows in each cycle) and thus $\widetilde{\mathcal{O}}(|\Gamma| + |U|)$ time.

For the insertion of U_i into Γ, we distinguish two cases. The first one (lines 7 to 10) is rather easy: If there is no node in Γ with the same value as U_i, we can just insert U_i and adjust the bit vector using the comparision procedure from 5. No marking involving U_i (this includes Λ markings of other nodes in U) needs to be changed. The second case (lines 11 to 23) is somewhat more complicated. Here we have $\varepsilon(U_i) = \varepsilon(V_j)$. The general idea is to remove U_i and use V_j instead in all markings M having U_i in their support. However, this does not work when $M(U_i) = M(V_j)$ (M would become doubly marked or target of a double arc). In that case, we remove both U_i and V_j from M and replace them by a node R_1 with value $2 \cdot \varepsilon(P_j)$. If again, R_1 becomes doubly marked by M, repeat (lines 19 to 23). This continues until we reach a node unmarked by M (or marked with the opposite sign) or when the sequence of nodes starting at P_j and each node having double the value of its predecessor (we call such a sequence a chain) ends. In order to cope with the latter, we create a new node R_k (which is, of course, completely unmarked) in lines 12 to 18. Doubling a node is done by increasing its Λ-marking by one in the obvious way. An extension of the chain starting at the 1-node might be needed, so we create that first (new node B).

Now, let us look at the time complexity. Topological ordering of U takes linear time in the number of arcs which is bounded by $|U| \cdot (|\Gamma| + |U|)$. For the analysis of the main loop, let us first ignore lines 19 to 23. Apart from the $\log(|\Gamma| + |U|)$ comparisons used in the binary search (each $\mathcal{O}(|\Gamma|$ time), we only need a constant number of comparisons and $\mathcal{O}(|\Gamma| + |U|)$ time e.g. for going through the bit vector. This is $\widetilde{\mathcal{O}}((|\Gamma| + |U|))$ per iteration.

Now to lines 19 to 23: As for the adjustment of markings, note that in every cycle of the inner while loop, we lose one mark or one edge originating in U. This adds up to $\mathcal{O}((|\Gamma| + |U|) \cdot |U| + |\sigma(M_1)| + \ldots + |\sigma(M_m)|)$. Note that markings having their support entirely in Γ never have to be altered during the whole process. ◀

▶ **Corollary 7.** *There is a* $\widetilde{\mathcal{O}}(|\Gamma|^2 + \sum_{M \in \mathcal{M}} |\sigma(M)|)$ *time procedure* Reduce *that reduces a power circuit* $\Pi = (\hat{\Gamma}, \hat{\delta})$ *with markings* $\mathcal{M}$*. The number of nodes at most triples. This is a special case of Theorem 6 where* $\Gamma = \emptyset$ *and* $U = \hat{\Gamma}$*.* ◀

▶ **Corollary 8.** *As a by-product, the procedure* Reduce *tests whether a dag* (Γ, δ) *defines a power circuit (i.e., all markings evaluate to integers).* ◀

By introducing a more sophisticated data structure, one gets rid of the log factors in the time complexity of ExtendReduction and Reduce, see [7]. Note that quadratic time is optimal, since the the number of arcs can be quadratic in the number of nodes.

3 Arithmetic in the semi-direct product $\mathbb{Z}[1/2] \rtimes \mathbb{Z}$

The basic data structure for this paper deals with the semi-direct product $\mathbb{Z}[1/2] \rtimes \mathbb{Z}$. Here $\mathbb{Z}[1/2]$ denotes the ring of rational numbers with denominators in $2^{\mathbb{N}}$ (It is also known as the ring of *dyadic rationals*.) Thus, an element in $\mathbb{Z}[1/2]$ is a rational number r which can be written as $r = u2^x$ with $u, x \in \mathbb{Z}$. We view $\mathbb{Z}[1/2]$ as an abelian group with addition. Multiplication by 2 defines an automorphism of $\mathbb{Z}[1/2]$, and hence the semi-direct product $\mathbb{Z}[1/2] \rtimes \mathbb{Z}$ becomes a (non-commutative) group where elements are pairs $(r, m) \in \mathbb{Z}[1/2] \times \mathbb{Z}$ and with the following explicit formulae for multiplication and inverses:

$$(r, m) \cdot (s, n) = (r + 2^m s, m + n), \qquad (r, m)^{-1} = (-r2^{-m}, -m)$$

The group $\mathbb{Z}[1/2] \rtimes \mathbb{Z}$ is isomorphic to the Baumslag-Solitar group $\mathbf{BS}(1, 2) = \langle a, t \mid tat^{-1} = a^2 \rangle$ via the mapping $a \leftrightarrow (1, 0), t \leftrightarrow (0, 1)$.

A sequence of s group operations may lead to exponentially large or exponentially small values in the first component. Binary representation can cope with these values. We equip $\mathbb{Z}[1/2] \rtimes \mathbb{Z}$ with a partially defined *swap operation*. For $(r, m) \in \mathbb{Z} \times \mathbb{Z} \subseteq \mathbb{Z}[1/2] \rtimes \mathbb{Z}$ we define $\mathrm{swap}(r, m) := (m, r)$. This looks innocent, but note that a sequence of $2^{\mathcal{O}(n)}$ defined operations starting with $(1, 0)$ may yield a pair $(0, \tau(n))$ where τ is the tower function. Indeed $\mathrm{swap}(1, 0) = (0, 1) = (0, \tau(0))$ and

$$\mathrm{swap}((0, \tau(n))(1, 0)(0, -\tau(n))) = \mathrm{swap}(\tau(n + 1), 0) = (0, \tau(n + 1)). \tag{1}$$

We also use triples to denote elements in $\mathbb{Z}[1/2] \rtimes \mathbb{Z}$. A triple $[u, x, k]$ with $u, x, k \in \mathbb{Z}$ and $x \leq 0 \leq k$ denotes the pair $(u2^x, k + x) \in \mathbb{Z}[1/2] \rtimes \mathbb{Z}$. For each element in $\mathbb{Z}[1/2] \rtimes \mathbb{Z}$ there are infinitely many corresponding triples. Using the generators a and t of $\mathbf{BS}(1, 2)$ one can write:

$$\begin{aligned} [u, x, k] = (u2^x, k + x) &= (0, x)(u, k) \in \mathbb{Z}[1/2] \rtimes \mathbb{Z} \\ &= t^x a^u t^k \in \mathbf{BS}(1, 2) \text{ and} \\ [u, x, k] \cdot [v, y, \ell] &= [u2^{-y} + v2^k, x + y, k + \ell] \end{aligned}$$

In what follows we use power circuits with *triple markings* for elements in $\mathbb{Z}[1/2] \rtimes \mathbb{Z}$. For $T = [U, X, K]$, where U, X, K are markings in a power circuit with $\varepsilon(U) = u$, $\varepsilon(X) = x \leq 0$, $\varepsilon(K) = k \geq 0$, we define $\varepsilon(T) \in Z[1/2] \rtimes \mathbb{Z}$ to be the triple $\varepsilon(T) = [u, x, k] = (u2^x, x + k)$.

Let us note that the Word Problem of $(\mathbb{Z}[1/2] \rtimes \mathbb{Z}, \cdot, \mathrm{swap})$ is solvable in polynomial time [7].

4 Solving the Word Problem in the Baumslag group

The Baumslag group[1] $G_{(1,2)}$ is a one-relator group with two generators a and b and the defining relation $a^{a^b} = a^2$. (The notation g^h means conjugation, here $g^h = hgh^{-1}$. Hence $a^{a^b} = bab^{-1}aba^{-1}b^{-1}$.) The group $G_{(1,2)}$ can be written as an HNN extension of $\mathbf{BS}(1,2) \simeq \mathbb{Z}[1/2] \rtimes \mathbb{Z}$ with *stable letter* b; and $\mathbf{BS}(1,2)$ is an HNN extension of $\mathbb{Z} \simeq \langle a \rangle$ with *stable letter* t:

$$\begin{aligned}\langle a, b \mid a^{a^b} = a^2 \rangle &\simeq \langle a, t, b \mid a^t = a^2, a^b = t \rangle \simeq \mathrm{HNN}(\langle a, t \mid a^t = a^2 \rangle, b, \langle a \rangle \simeq \langle t \rangle) \\ &\simeq \mathrm{HNN}\left(\mathrm{HNN}(\langle a \rangle, t, \langle a \rangle \simeq \langle a^2 \rangle), b, \langle a \rangle \simeq \langle t \rangle\right)\end{aligned}$$

Before the work of Myasnikov, Ushakov and Won [19], $G_{(1,2)}$ had been a possible candidate for a one-relator group with an extremely hard (non-elementary) word problem in the worst case by the result of Gersten [9]. (Indeed, the tower function occurs as follows: Let $T(0) = t$ and $T(n+1) = bT(n)aT(n)^{-1}b^{-1}$. Then $T(n) = t^{\tau(n)}$ by a translation of Equation 1.) The purpose of this section is to improve the $\mathcal{O}(n^7)$ time-estimation of [19] to (quasi-)cubic time. Theorem 9 yields the first practical algorithm to solve the Word Problem in $G_{(1,2)}$ for a worst-case scenario[2]. It has been implemented and runs in reasonable time on instances of several thousand letters.

▶ **Theorem 9.** *The Word Problem of the Baumslag group $G_{(1,2)}$ is decidable in time $\widetilde{\mathcal{O}}(n^3)$.*

Proof. We assume that the input is already in compressed form, given by a sequence of letters b, b^{-1} and triple markings $[U, X, K]$, the latter representing elements in $\mathbb{Z}[1/2] \rtimes \mathbb{Z}$, which in turn encode words over $a^{\pm 1}$'s and $t^{\pm 1}$'s. We use the following invariants:

i) U, X, K have pairwise disjoint supports.
ii) U is a source.
iii) All incoming arcs to $X \cup K$ have their origin in U.
iv) Arcs from U to X have the opposite sign of the corresponding node-sign in X.

These are clearly satisfiable in case we start with a sequence of $a^{\pm 1}$'s, $t^{\pm 1}$'s, and $b^{\pm 1}$'s (e.g. create disjoint power circuits, one for each marking, as in Example 2). The formula $[u, x, k] \cdot [v, y, \ell] = [u2^{-y} + v2^k, x + y, k + \ell]$ allows to multiply elements in $\mathbb{Z}[1/2] \rtimes \mathbb{Z}$ without destroying the invariants or increasing the total number of nodes in the power circuits (the invariants make sure that cloning is not necessary). The total number of multiplications is bounded by n. Taking into account that there are at most n^2 arcs, we are within the time bound $\mathcal{O}(n^3)$.

Now we perform leftmost Britton reductions, see [15]. In terms of group generators this means replacing factors ba^sb^{-1} by t^s and $b^{-1}t^sb$ by a^s (always replacing the leftmost occurance first). Thus, if we see a subsequence $b[u, x, k]b^{-1}$, then we must check if $x + k = 0$ and after that if $u2^x \in \mathbb{Z}$. If we see a subsequence $b^{-1}[u, x, k]b$, then we must check $u = 0$. In the positive cases we swap, in the other case we do nothing. Let us give the details: For a test we reduce a copy of the circuit using REDUCE which takes time $\widetilde{\mathcal{O}}(n^2)$. After each test for a Britton reduction, the copy is deleted. There are two possibilities for necessary tests.

1.) $u = 0$. If yes, remove in the original power circuit the source U, this makes $X \cup K$ a source; replace $[u, x, k]$ by $[x + k, 0, 0]$. The invariants are satisfied.

[1] sometimes called Baumslag-Gersten group, not to be confused with the Baumslag-Solitar group $\mathbf{BS}(1,2)$
[2] It is easy to design simple algorithms which perform extremely well on random inputs. But all these algorithms fail on short instances, e.g. in showing $tT(6) = T(6)t$.

2.) $x + k = 0$. If yes, check whether $u2^x \in \mathbb{Z}$. If yes, replace $[u, x, k]$ in the original power circuit by either $[0, u2^x, 0]$ or $[0, 0, u2^x]$ depending on whether $u2^x$ is negative or positive. We get $u2^x$ without increasing the number of nodes, since arcs from U to X have the opposite signs of the node-signs in X. Thus, if E has been the set of arcs before the test, it is switched to $U \times X \setminus E$ after the test. The new marking for $u2^x$ is a source and does not introduce any cycle, because its support is still the support of the source U.

It is easy to see that computing a Britton reduction on an input sequence of size n, we need at most $2n$ tests and at most n of them are successful. Hence we are still within the time bound $\widetilde{\mathcal{O}}(n^3)$. ◀

5 Solving the Word Problem in Higman's group H_4

The Higman group H_q has a finite presentation with generators $a_1, \ldots, a_q$ and defining relations $a_p a_{p-1} a_p^{-1} = a_{p-1}^2$ for all $p \in \mathbb{Z}/q\mathbb{Z}$. It is known [27] that H_q is trivial for $q \leq 3$ and infinite for $q \geq 4$. From now on, we focus on the group H_4 which is the one usually referred to as *the* Higman group. This group was the first example of a finitely generated group where all finite quotient groups are trivial. It was another potential natural candidate for a group with an extremely hard (non-elementary) word problem in the worst case. Indeed, define:

$$w(p, 0) = a_p \ (p \in \mathbb{Z}/4\mathbb{Z}), \qquad w(p-1, i+1) = w(p, i) a_{p-1} w(p, i)^{-1} \ (i \in \mathbb{N}, p \in \mathbb{Z}/4\mathbb{Z})$$

By induction, $w(p, n) = a_p^{\tau(n)} \in H_4$, where $\tau(n)$ is the n-th value of the tower function, but the length of the words $w(p, n)$ is $2^{n+1} - 1$, only. Hence there is a "tower-sized gap" between input length and length of a canonical normal form.

We define the group G_{12} by the generators a_1 and a_2, and defininig relation $a_2 a_1 a_2^{-1} = a_1^2$. In the same way we define G_{23}, G_{34}, and G_{41}. As we saw in Section 3, each of these groups is isomorphic to the Baumslag-Solitar group $\mathbf{BS}(1, 2)$ and to $\mathbb{Z}[1/2] \rtimes \mathbb{Z}$. Furthermore, we define the group G_{123} by the generators a_1, a_2, a_3 and defining relations $a_2 a_1 a_2^{-1} = a_1^2$ and $a_3 a_2 a_3^{-1} = a_2^2$. (Similarly define G_{341}.) We have $G_{123} = G_{12} *_{F_2} G_{23}$ where F_2 is the free subgroup of both G_{12} and G_{23} generated by a_2. Finally, we get H_4 as an amalgamated product

$$H_4 \simeq G_{123} *_{F_{13}} G_{341},$$

where F_{13} is the free subgroup of rank two of G_{123} and G_{341} with basis $\{a_1, a_3\}$, see [27].

This isomorphism yields a direct proof that H_4 is an infinite group, see [27]. In the following we use some well-known facts about amalgamated products, see [15, 27, 6]. In order to solve the Word Problem, we start with an alternating sequence of group elements from G_{123} and G_{341}. The sequence can be shortened, only if one factor happens to be in the subgroup F_{13}. In this case we swap the factor from G_{123} to G_{341} and vice versa. By abuse of language we call this procedure again a *Britton reduction.* (This is perhaps not standard notation, but it conveniently unifies the same phenomenon in amalgamated products and HNN-extensions.) A sequence evaluating to 1 in H_4 can be entirely eliminated using these kinds of Britton reductions.

Elements in the groups $G_{i,i+1}$ ($i \in \mathbb{Z}/4\mathbb{Z}$) are represented by triple markings $T = [U, X, K]$ in some power circuit. In order to remember that we evaluate T in the group $G_{i,i+1}$, we give each T a *type* $(i, i+1)$, which is recorded as a subscript. For $\varepsilon(T) = [u, x, k]$ we obtain:

$$\begin{aligned} \varepsilon(T_{(i,i+1)}) &= a_{i+1}^x a_i^u a_{i+1}^k && \in G_{i,i+1} \\ &= a_i^{u2^x} a_{i+1}^{x+k} && \text{if } u2^x \in \mathbb{Z} \end{aligned}$$

From now on we work with a single power circuit Π together with a sequence $(T_j)_{j\in J}$ of triple markings of various types. This is given as a tuple $\mathcal{T} = (\Gamma, \delta; (T_j)_{j\in J})$. If (Γ, δ) is reduced, then $\mathcal{T}$ is called a *main data structure* .

▶ **Definition 10.** The *weight* $\omega(T)$ of a triple marking $T = [U, X, K]$ is defined as $\omega(T) = |\sigma(U)| + |\sigma(X)| + |\sigma(K)|$. The *weight* $\omega(\mathcal{T})$ of a main data structure $\mathcal{T}$ is defined as $\omega(\mathcal{T}) = \sum_{j\in J} \omega(T_j)$. Its *size* $\|\mathcal{T}\|$ is defined by $\|\mathcal{T}\| = |\Gamma|$.

The following basic operations are defined on a main data structure. Applying a basic operation means replacing the left-hand side of the equation by the right-hand side, thus forgetting any markings of the replaced triple(s). To improve readability, we write them down only for G_{123}, but they are also defined for G_{341}.

- Multiplication: $[u, x, k]_{(1,2)} \cdot [v, y, \ell]_{(1,2)} = [u2^{-y} + v2^k, x + y, k + \ell]_{(1,2)}$
- Swapping from $(1, 2)$ to $(2, 3)$: $[0, x, k]_{(1,2)} = [x + k, 0, 0]_{(2,3)}$
- Swapping from $(2, 3)$ to $(1, 2)$: $[z, 0, 0]_{(2,3)} = \begin{cases} [0, 0, z]_{(1,2)} & \text{for } z \geq 0 \\ [0, z, 0]_{(1,2)} & \text{for } z < 0. \end{cases}$
- Splitting: $[u, x, k]_{(1,2)} = [u2^x, 0, 0]_{(1,2)} \cdot [0, x, k]_{(1,2)}$ for $u2^x \in \mathbb{Z}$
  $[u, x, k]_{(2,3)} = [0, x, k]_{(2,3)} \cdot [u2^{-k}, 0, 0]_{(2,3)}$ for $u2^{-k} \in \mathbb{Z}$
  We allow splitting operations only when immediatelly followed by a multiplication, thus we never increase the number of triple markings inside $\mathcal{T}$.

We keep $\mathcal{T}$ as a main data structure by doing addition and multiplication by powers of 2 using clones (as described in Section 2) and calling ExtendReduction on these after each basic operation.

▶ **Proposition** 11. Let $\mathcal{T} = (\Gamma, \delta; (T_j)_{j\in J})$ be a main data structure of size at most m, weight at most w (and with $|J| + w \leq m$). The following assertions hold.

i) No basic operation increases the weight of $\mathcal{T}$.
ii) Each basic operation increases the size $\|\mathcal{T}\|$ by $\mathcal{O}(w)$.
iii) Each basic operation takes time $\widetilde{\mathcal{O}}(mw)$.
iv) A sequence of s basic operations takes time $\widetilde{\mathcal{O}}(s^2m^2)$ and the size of $\mathcal{T}$ remains bounded by $\mathcal{O}(m + sw)$.

Proof. We can do the necessary tests, because the power circuit is reduced (Proposition 5). Before each operation we replace the involved markings in $(T_j)_{j\in J}$ by clones, which increases the size by $\mathcal{O}(w)$, but does not increase the weight. Note that there is enough time to create the clones with all their outgoing arcs. With the new clones we can perform the operations by using the algorithms described in Section 2. We regain the main data structure by calling ExtendReduction which integrates the modified clones into the reduced representation.

In order to get iv), we observe that the final size of the circuit is bounded by $\mathcal{O}(m + sw)$, so we need at most $s \cdot \widetilde{\mathcal{O}}((m + sw) \cdot w) \subseteq \widetilde{\mathcal{O}}(s^2m^2)$ time for all s operations. ◀

▶ **Theorem 12.** *The Word Problem of H_4 can be solved in time $\widetilde{\mathcal{O}}(n^6)$.*

The traditional input for a Word Problem solving algorithm is a word over the generators a_p and their inverses a_p^{-1}. We solve a slightly more general problem by assuming that the input consists of a power circuit $\Pi = (\Gamma, \delta)$ together with a sequence of s triple markings of various types. Each triple marking $[U, X, K]_{(p,p+1)}$ corresponds to $a_{p+1}^{\varepsilon(X)} a_p^{\varepsilon(U)} a_{p+1}^{\varepsilon(K)} \in H_4$.

Let w be the total weight of $\mathcal{T} = (\Gamma, \delta; (T_j)_{1\leq j\leq s})$. For simplicity we assume $s \leq w$ and that w and sizes of clones are bounded by $\|\mathcal{T}\| = |\Gamma|$. Having $s \leq w \leq n \in \mathcal{O}(w)$, we can think of $n = |\Gamma|$ as our input size. We transform $\mathcal{T}$ into a main data structure by invoking Reduce.

During the procedure $|\Gamma|$ increases, but the number of triple markings remains bounded by s and the weight by w. In order to achieve our main result we show how to solve the word problem with $\mathcal{O}(s^2)$ basic operations on the main data structure $\mathcal{T}$. Assuming this, by Proposition 11, the final size will be bounded by $m \in \mathcal{O}(s^2 w)$; and the time for all basic operations is $\widetilde{\mathcal{O}}(s^4 w^2) \subseteq \widetilde{\mathcal{O}}(n^6)$.

We collect sequences of triple markings of type $(1,2)$ and $(2,3)$ in *intervals* $\mathcal{L}$, which in turn receive type $(1,2,3)$; and we collect triple markings of type $(3,4)$ and $(4,1)$ in intervals of type $(3,4,1)$. Each interval has (as a sequence of triple markings) a semantics $\varepsilon(\mathcal{L})$ which is a group element either in G_{123} or in G_{341} depending on the type of $\mathcal{L}$. Thus, it makes sense to ask whether $\varepsilon(\mathcal{L}) \in F_{13}$. These tests are crucial and dominate the runtime of the algorithm.

Now the sequence $(T_j)_{1 \le j \le s}$ of triple markings appears as a sequence of intervals:

$$(\mathcal{L}_1, \ldots, \mathcal{L}_f; \mathcal{L}_{f+1}, \ldots, \mathcal{L}_t).$$

We introduce a separator ";" dividing the list in two parts. The following invariants are kept up:

i) All $\mathcal{L}_1, \ldots, \mathcal{L}_f$ satisfy $\varepsilon(\mathcal{L}_i) \notin F_{13}$. In particular, these intervals are not empty and they represent non-trivial group elements in $(G_{123} \cup G_{341}) \setminus F_{13}$.
ii) The types of intervals left of the separator are alternating.

In the beginning each interval consists of exactly one triple marking, thus $f = 0$ and $t = s$. The algorithm will stop either with $1 \le f = t$ or with $f = 0$ and $t = 1$.

Now we describe how to move forward: First, assume $f = 0$ (thus $t > 1$). If $\varepsilon(\mathcal{L}_1) \notin F_{13}$, then move the separator to the right, i.e. we obtain $f = 1$. If $\varepsilon(\mathcal{L}_1) \in F_{13}$, then, after swapping $\mathcal{L}_1$, we can join the intervals $\mathcal{L}_1$ and $\mathcal{L}_2$. In this case we still have $f = 0$, but t decreases by 1.

Now assume that $0 < f < t$. If $\mathcal{L}_f$ and $\mathcal{L}_{f+1}$ have the same type, then append $\mathcal{L}_{f+1}$ to $\mathcal{L}_f$, and move the separator to the left of $\mathcal{L}_f$. Thus, f and t decrease by 1.

If $\mathcal{L}_f$ and $\mathcal{L}_{f+1}$ have different types, then we test whether $\varepsilon(\mathcal{L}_{f+1}) \in F_{13}$. If $\varepsilon(\mathcal{L}_{f+1}) \notin F_{13}$, then move the separator to the right, i.e. f increases by 1. If $\varepsilon(\mathcal{L}_{f+1}) \in F_{13}$, then we swap $\mathcal{L}_{f+1}$ and join the intervals $\mathcal{L}_f$ and $\mathcal{L}_{f+1}$ into one new interval. Since we do not know whether the new interval belongs to F_{13}, we put the separator in front of it, decreasing both f and t by 1.

If we terminate with $1 \le f = t$, then $\varepsilon(\mathcal{L}_1) \cdots \varepsilon(\mathcal{L}_t) \in H_4$ is a Britton-reduced sequence in the amalgamated product. It represents a non-trivial group element, since $t \ge 1$.

In the other case we terminate with $f = 0$ and $t = 1$. We will make sure that the subgroup membership test for F_{13} can as a by-product also answer the question whether a sequence $\varepsilon(\mathcal{L})$ represents the trivial group element. If we do so, one more test on $(\mathcal{L}_1)$ yields the answer we need.

The number of membership tests for F_{13} is bounded by $2s$. All that remains, is to prove:

▶ **Lemma 13.** *The test for membership of $\varepsilon(\mathcal{L})$ in the subgroup F_{13} can be realized with $\mathcal{O}(s)$ basic operations in the main data structure $\mathcal{T}$. The test yields either "no" or it says "yes" with the additional information whether or not $\varepsilon(\mathcal{L})$ is the trivial group element. Moreover, in the "yes" case we can also swap the type of $\mathcal{L}$ within the same bound on basic operations.*

Proof. Assume that $\mathcal{L}$ is of type $(1,2,3)$, so it contains only triples of types $(1,2)$ and $(2,3)$. Let s be the length of $\mathcal{L}$. In a first round we create a sequence of triple markings $(T_1, \ldots, T_t)$ with $t \le s$ such that for $1 \le i < t$ the type of T_i is $(1,2)$ if and only if the type of T_{i+1} is

$(2,3)$. We can do so by $s-t$ multiplications from left to right without changing the semantics of $g = \varepsilon(T_1)\cdots\varepsilon(T_t) \in G_{123}$.

Next, we make this sequence Britton-reduced (which also gives us the information whether the sequence represents the identity in the group). Again, we scan from left to right. If we are at $T = T_i$ with value $[u,x,k]$ we have to check whether either $[u,x,k]_{(1,2)} = (0,z) \in \mathbb{Z}[1/2]\rtimes\mathbb{Z}$ or $[u,x,k]_{(2,3)} = (z,0) \in \mathbb{Z}[1/2]\rtimes\mathbb{Z}$ for some integer $z \in \mathbb{Z}$.

For the type $(1,2)$ we have $[u,x,k]_{(1,2)} = (0,z)$ if and only if $u = 0$, which in a reduced circuit means that the support of the marking for u is empty. Hence this test is trivial. If the test is positive, we can replace $[u,x,k]_{(1,2)}$ by $[0,x,k]_{(1,2)}$ and perform a swap to type $(2,3)$. If $t > 1$ we perform multiplications with its neighbors, thereby decreasing the value t.

For the type $(2,3)$ we have $[u,x,k]_{(2,3)} = (z,0)$ if and only if both $k+x = 0$ and $u2^x \in \mathbb{Z}$. These tests are possible in linear time and if successful, we continue as in the precedent case.

The final steps are more subtle. Let $\varepsilon(T_j) = g_j \in G_{12}\cup G_{23}$. Recall that $(g_1,\ldots,g_t)$ is already a Britton-reduced sequence. We have $g_1\cdots g_t \in F_{13}$ if and only if there is a sequence $(h_0,h_1,\ldots,h_t)$ with the following properties:

i) $h_0 = h_t = 1$ and $h_j \in \langle a_2\rangle$ for all $0 \le j \le t$.
ii) $h_{j-1}g_j = g'_j h_j$ with $g'_j \in F_1 \cup F_3$ for all $1 \le j \le t$.

Assume that such a sequence $(h_0,h_1,\ldots,h_t)$ exists. Then we have $g'_j \in \langle a_1\rangle$ if and only if $g_j \in G_{12}$. Moreover, whenever $gh = g'h' \in G_{123}$ with $g,g' \in F_1\cup F_3$ and $h,h' \in \langle a_2\rangle$, then $g = g'$ and $h = h'$. This follows because $g'^{-1}g = h'h^{-1} \in F_{13}\cap\langle a_2\rangle = \{1\}$. Thus, the product $h_{j-1}g_j$ uniquely defines $g'_j \in F_1\cup F_3$ and $h_j \in \langle a_2\rangle$, because $h_0 = 1$ is fixed.

The invariant during a computation from left to right is that $\varepsilon(T_j) = h_{j-1}g_j$. We obtain $\varepsilon(T_j) = g'_j h_j$ by a splitting operation. If no splitting is possible we know that $g \notin F_{13}$ and we can stop. If, however, a splitting is possible, then we have two cases. If j is the last index ($j = t$), then, in addition, we must have $h_j = 1$. We can test this. If the test fails, we stop with $g \notin F_{13}$. If we are not at the last index we perform a swap of the right-hand factor and multiply it with the next triple marking, which has the correct type to do so. As our sequence has been Britton-reduced, the total number of triple markings remains constant. There can be no cancelations at this point. Thus, the test gives us the answer to the subgroup membership problem using $\mathcal{O}(s)$ basic operations. ◀

6 Conclusion and future research

The Word Problem is a fundamental problem in algorithmic group theory. In some sense "almost all" finitely presented groups are hyperbolic [22] and satisfy a "small cancelation" property, so the Word Problem is solvable in linear time! For hyperbolic groups there are also efficient parallel algorithms and the Word Problem is in $\mathbf{NC^2}$ [4]. On the other hand, for many naturally defined groups little is known. Among one-relator groups the Baumslag group $G_{(1,2)}$ was supposed to have the hardest Word Problem [19], but we solved it in cubic time. The method generalizes to the higher Baumslag groups $G_{(m,n)}$ in case that m divides n, but this requires more "power circuit machinery" and has not been worked out in full details yet, see [19]. The situation for $G_{(2,3)}$ is open and related to questions in number theory. Baumslag and Higman groups are built via HNN extensions and amalgamated products. Many algorithmic problems are open for such constructions, for advances about theories of HNN-extensions and amalgamated products we refer to [14].

Another interesting open problem concerns the Word Problem in *Hydra* groups. Doubled hydra groups have Ackermannian Dehn functions [8], but still it is possible that their Word Problem is solvable in polynomial time.

References

1 G. Baumslag. A non-cyclic one-relator group all of whose finite quotients are cyclic. *J. Austr. Math. Soc.*, 10(3-4):497–498, 1969.

2 G. Baumslag, A. G. Myasnikov, and V. Shpilrain. Open problems in combinatorial group theory. Second Edition. In *Combinatorial and geometric group theory*, volume 296 of *Contemporary Mathematics*, pages 1–38. American Mathematical Society, 2002.

3 W. W. Boone. The Word Problem. *Annals of Mathematics*, 70(2):207–265, 1959.

4 J.-Y. Cai. Parallel computation over hyperbolic groups. In *Proc. 24th ACM Symp. on Theory of Computing, STOC 92*, pages 106–115. ACM-press, 1992.

5 M. Dehn. Über unendliche diskontinuierliche Gruppen. *Math. Ann.*, 71(1):116–144, 1911.

6 V. Diekert, A. J. Duncan, and A. G. Myasnikov. Geodesic rewriting systems and pregroups. In O. Bogopolski et al. (eds.), *Combinatorial and Geometric Group Theory*, Trends in Mathematics, pages 55–91. Birkhäuser, 2010.

7 V. Diekert, J. Laun, and A. Ushakov. Efficient algorithms for highly compressed data: The Word Problem in Higman's group is in P. *ArXiv e-prints*, Mar. 2011.

8 W. Dison and T. R. Riley. Hydra groups. *ArXiv e-prints*, abs/1002.1945, Feb. 2010.

9 S. M. Gersten. Isodiametric and isoperimetric inequalities in group extensions. 1991.

10 G. Higman. A finitely generated infinite simple group. *J. LMS*, 26:61–64, 1951.

11 I. Kapovich, A. G. Miasnikov, P. Schupp, and V. Shpilrain. Generic-case complexity, decision problems in group theory and random walks. *J. Algebra*, 264:665–694, 2003.

12 M. Lohrey. Word problems and membership problems on compressed words. *SIAM J. Comput.*, 35(5):1210–1240, 2006.

13 M. Lohrey and S. Schleimer. Efficient computation in groups via compression. In V. Diekert et al. (eds.), *CSR*, *LNCS*, 4649:249–258. Springer, 2007.

14 M. Lohrey and G. Sénizergues. Theories of HNN-extensions and amalgamated products. In M. Bugliesi et al. (eds.), *ICALP*, *LNCS*, 4052:504–515. Springer, 2006.

15 R. Lyndon and P. Schupp. *Combinatorial Group Theory*. Springer, 2001.

16 K. Madlener and F. Otto. Pseudo-natural algorithms for finitely generated presentations of monoids and groups. *J. Symb. Comput.*, 5:339–358, 1988.

17 W. Magnus. Das Identitätsproblem für Gruppen mit einer definierenden Relation. *Math. Ann.*, 106:295–307, 1932.

18 A. G. Myasnikov, A. Ushakov, and D. W. Won. Power circuits, exponential algebra, and time complexity. *ArXiv e-prints*, abs/1006.2570, 2010. To appear in IJAC.

19 A. G. Myasnikov, A. Ushakov, and D. W. Won. The Word Problem in the Baumslag group with a non-elementary Dehn function is polynomial time decidable. *Journal of Algebra*, 345(1):324–342, 2011.

20 A. G. Myasnikov and S. Ushakov. Cryptography And Groups (CRAG). Software Library.

21 P. S. Novikov. On the algorithmic unsolvability of the word problem in group theory. *Trudy Mat. Inst. Steklov*, pages 1–143, 1955. In Russian.

22 A. Y. Ol'shanskii. Almost every group is hyperbolic. *Int. J. Alg. Comp.*, 2:1–17, 1992.

23 F. Otto, D. E. Cohen, and K. Madlener. Separating the intrinsic complexity and the derivational complexity of the word problem for finitely presented groups. *Math. Log. Q.*, 39:143–157, 1993.

24 A. N. Platonov. Isoparametric function of the Baumslag-Gersten group. *Vestnik Moskov. Univ. Ser. I Mat. Mekh.*, 3:12–17, 2004. Russian. Engl. transl. Moscow Univ. Math. Bull. 59 (3) (2004), 12–17.

25 M. V. Sapir, J.-C. Birget, and E. Rips. Isoperimetric and Isodiametric Functions of Groups. *Ann. Math.*, 156:345–466, 2002.

26 S. Schleimer. Polynomial-time word problems. *Comm. Math. Helv.*, 83:741–765, 2008.

27 J.-P. Serre. *Trees*. Springer, 1980.

Efficiently Decodable Compressed Sensing by List-Recoverable Codes and Recursion*

Hung Q. Ngo[1], Ely Porat[2], and Atri Rudra[1]

1 **Department of Computer Science and Engineering**
University at Buffalo, SUNY,
Buffalo, NY, 14260.
{hungngo,atri}@buffalo.edu

2 **Department of Computer Science,**
Bar-Ilan University,
Ramat Gan 52900, Israel.
porately@cs.biu.ac.il

Abstract

We present two recursive techniques to construct compressed sensing schemes that can be "decoded" in *sub-linear* time. The first technique is based on the well studied code composition method called *code concatenation* where the "outer" code has strong *list recoverability* properties. This technique uses only one level of recursion and critically uses the power of list recovery. The second recursive technique is conceptually similar, and has multiple recursion levels. The following compressed sensing results are obtained using these techniques:

- (*Strongly explicit efficiently decodable ℓ_1/ℓ_1 compressed sensing matrices*) We present a *strongly explicit* ("for all") compressed sensing measurement matrix with $O(d^2 \log^2 n)$ measurements that can output near-optimal d-sparse approximations in time $\text{poly}(d \log n)$.
- (*Near-optimal efficiently decodable ℓ_1/ℓ_1 compressed sensing matrices for non-negative signals*) We present two randomized constructions of ("for all") compressed sensing matrices with near optimal number of measurements: $O(d \log n \log \log_d n)$ and $O_{m,s}(d^{1+1/s} \log n (\log^{(m)} n)^s)$, respectively, for any integer parameters $s, m \geq 1$. Both of these constructions can output near optimal d-sparse approximations for *non-negative* signals in time $\text{poly}(d \log n)$.

To the best of our knowledge, none of the results are dominated by existing results in the literature.

1998 ACM Subject Classification F.2.1 Computations on Matrices, E.4. Coding and Information Theory

Keywords and phrases Compressed Sensing, Sub-Linear Time Decoding, List-Recoverable Codes

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.230

1 Introduction

Compressed sensing [4,7] is a sparse recovery problem that has seen a surge in recent research activity due to a wide variety of practical applications [8]. (The literature is vast: we will only refer to the very closely related work in this paper. The reader is referred to the survey [8] and the references therein for more details.) Compressed sensing (CS) has two

* E. Porat's research was partially supported by BSF grant 2006334 and ISF grant 1484/08 while A. Rudra's research was supported by NSF CAREER grant CCF-0844796.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 230–241

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

components. The combinatorial part is to design a $t \times N$ *measurement matrix* M (where typically $t \ll N$) such that, given the "measurements" Mx of any signal $x \in \mathbb{R}^N$ one needs to recover a sparse approximation of x. More precisely the algorithmic task is as follows. Given the measurements $y = Mx + \nu$ which was contaminated with a noise vector ν, the ℓ_p/ℓ_p CS problem is to compute a vector $\hat{x} \in \mathbb{R}^N$ (ideally $\hat{x}$ is d-sparse or $O(d)$-sparse, i.e. having at most d or $O(d)$ non-zero entries for some parameter $1 \leq d \ll N$) such that the following conditions holds: $\|x - \hat{x}\|_p \leq C \cdot \|x - x_d^*\|_p + C' \cdot \|\nu\|_p$, where x_d^* is the vector x with all but its d highest-magnitude components zeroed out. In the above $C \geq 1$ is the approximation factor. Ideally, we would like to achieve $C = 1 + \epsilon$ for any $\epsilon > 0$. The noise dependency C' should also be as small as possible. Typically, we consider $p = 1$ in the "for all" case (i.e. the same matrix has to work for every signal) and $p = 2$ in the "for each" case (i.e. a distribution on matrices that works with high probability for each signal). This paper will concentrate on the ℓ_1/ℓ_1 for all problem.

The primary objective in compressed sensing is to minimize the number of measurements t. It is known that $t = \Theta(d \log(N/d))$ measurements are both necessary and sufficient [3]. The second objective is to "decode" (i.e. compute $\hat{x}$ given $y = Mx$) efficiently. It was shown recently that for the ℓ_1/ℓ_1 for all problem, decoding can be done in time $O(N \log N)$ while still attaining the optimal $O(d \log(N/d))$ number of measurements [14].

While near linear time decoding is fairly efficient, it is natural to wonder whether one could decode in *sub-linear* time. In particular, can we achieve decoding with time $\text{poly}(d, \log N)$? Note that when d is small, then $\text{poly}(d, \log N)$ could be an exponential improvement over $\tilde{O}(N)$. Given the wide applicability of compressed sensing, this is an interesting theoretical question in itself. In particular, compressed sensing is directly related to data streaming [16], where $\text{poly}(d, \log N)$-time decoding is crucial.

For the ℓ_1/ℓ_1 for all problem, sublinear time decoding for compressed sensing has been considered by Gilbert et al. [9]. They achieve a $\text{poly}(t)$ time decoding with a sub-optimal $t = O(d \log^c N)$ number of measurements (where $c \geq 4$ is some absolute constant). Their result has a few more shortcomings: (i) their measurement matrix M is randomized; and (ii) measurement noise was not handled. Indyk and Ruzic [14] overcome these drawbacks but they can only obtain near-linear time decoding. Very recently, Porat and Strauss [19] obtain the optimal number of measurements with sub-linear decoding time. However, the decoding time is always polynomial in N.

Our Results. We have three results for the sub-linear time decodable ℓ_1/ℓ_1 for all problem. (We clarify that without the sub-linear time decoding aspect, better results especially w.r.t. the number of measurements are known.) The first result is a CS matrix that uses $t = O(d^2 \log^2 N)$ measurements. This is an improvement over [9] only for $d = o(\log^2 N)$. However, our scheme has a couple of advantages: (i) the matrix M is *strongly explicit* (i.e. any entry in the matrix can be computed in $\text{poly}(\log N)$ time) and (ii) it can handle measurement noise. Our construction and decoding schemes are arguably much simpler: the matrix is the classic group testing matrix based on Reed-Solomon codes [15].

Our next two results only work for the case when the original signal x is non-negative (the first result works for arbitrary signals). While the constraint is restrictive, it does hold in some applications, such as when x is an image pixel vector, or when we address super-imposed coding problems under multiple-access *adder channels* [1]. The second result is a randomized CS scheme for non-negative signals with $t = O(d \log N \log \log_d N)$ measurements along with $\text{poly}(t)$ decoding time. However, this result cannot handle measurement noise. The third result is a randomized CS scheme for non-negative signals with $t = O(d^{1+1/s} \log N (\log^{(m)} N)^s)$, for any integer parameters $s, m \geq 1$ where we have suppressed some terms that depend

only on s and m and $\log^{(m)}(\cdot)$ denotes the $\log(\cdot)$ operator applied m times. Though the number of measurements is worse, this scheme can handle measurement noise and is efficiently decodable.

All of our CS results obtain an approximation ratio of $C = 1 + \epsilon$ for arbitrary $\epsilon > 0$, with the dependence of t on ϵ matching the best known quadratic guarantee of [14].

Our Techniques. The strongest point of our paper is probably its conceptual simplicity.

There is a generic decoding framework that has been used many times before: e.g. in [6, 9, 14] for CS and in [13, 17] for group testing. The decoding has two main steps. The first step, called *filtering*, approximately identifies the locations of the "heavy hitters" in x, whose omission from the output would likely result in a large approximation factor. The second step, called *estimation*, assigns values to these coordinates to obtain $\hat{x}$. The final CS matrix is obtained by vertically stacking the filtering and the estimation matrices.

The main insight in sublinear time decodable schemes in this paper (and in [13, 17]) is two-fold. The first observation is that many existing estimation algorithms (e.g., [9, 14, 19]) will work in (near) linear time in $|S|$ if given a set $S \subseteq [N]$ of coordinates which do not miss too much heavy hitter mass. Thus, to get a sublinear time decoding algorithm, it would be sufficient to compute S in the filtering stage with $|S| = \text{poly}(d, \log N)$ in time $\text{poly}(d, \log N)$. The second observation is that, by taking advantage of the fact that $|S|$ can be larger than d, we can design the filtering matrix with about $O(d \log N)$ measurements.

The main technical contribution of this paper is in the filtering step.

Let's start with the $O(d^{1+1/s} \log N (\log^{(m)} N)^s)$ ℓ_1/ℓ_1 for each result. In this case we use only one level of recursion. In particular, we use a $(n, k)_q$ code[1] to "hash" each of the N coordinates of x into nq "buckets" n times. (In particular, if the ith codeword in the jth position has a symbol β, where $i \in [N], j \in [n], \beta \in [q]$, then the jth copy of x_i goes to the (j, β) bucket.) Then we use another filtering scheme on each of the n chunks of q buckets corresponding to each position in the codeword. Three things are in our favor: (i) We can pick $q \ll N$, which means we can use a "wasteful" filtering scheme, such as the identity matrix, on each of n chunks of buckets; (ii) Since x is non-negative it is not too hard to show that a heavy hitter in x is likely "contained" in a heavy hitter of the hashed-down domain of size q; (iii) A simple Markov argument shows that if we pick a large enough number of heavy hitters in the domain of size q, then a lot of non-heavy hitters in x will not suddenly become heavy hitters in the smaller domain due to collisions with other non-heavy hitters in x. This implies that in sub-linear time, we can get for each of the n chunks of buckets, a small list of possible bucket locations that the heavy hitters in x will reside. (A similar construction was used for group testing by the authors in [17].)

In coding terminology, the remaining problem is the following: for every $i \in [n]$, given a small subset $S_i \subseteq [q]$, we want to output all codewords $(c_1, \ldots, c_n)$ such that $c_i \in S_i$ for every $i \in [n]$. Using a simple Markov argument one can get a similar result with measurement noise except we'll need to work with the weaker condition that $c_i \in S_i$ for at least $n/2$ values of $i \in [n]$. It turns out that this problem is *exactly* the problem of list recovery (cf. [10]). The recent work of Parvaresh and Vardy [18] leads to excellent list recoverable codes that can perform the task above algorithmically in time $\text{poly}(n)$ (which in our setting of parameters is $\text{poly}(t)$). However, these codes have too large a q for our purposes.

Fortunately, if we recursively combine several families of Parvaresh-Vardy codes (via a well-known code composition technique called "code concatenation"), then we get codes over

[1] I.e., we use a code with $N = q^k$ codewords each of which is a vector in $[q]^n$. See Section 2 for more details on coding terminology/definitions.

acceptably small q that still have acceptable list recoverability. Typically, code concatenation is done with two *different* families of codes while ours does it with the same family.

The $O(d \log N \log \log_d N)$ result follows by a different recursive construction which has multiple recursive levels. (Similar construction was again used for group testing by the authors in [17].) Here is the main idea behind the construction. Let us consider the simple case where by a simple hashing we map the N coordinates of x into two domains of size $\sqrt{N}$ each. The way we do this is hashing all coordinate indices that agree in the first $\log N/2$ bits into one bucket (and we similarly do this for the last $\log N/2$ bits). Now recursively we obtain sub-linear time decodable filtering schemes that work on domains of size $\sqrt{N}$. In other words, we will get two lists S_1 and S_2 which will contain the first and second $\log N/2$ bits of the indices of the heavy hitters of x respectively. Note that all the indices of the heavy hitters are contained in the set $S_1 \times S_2$. To complete the recursive step, we use a filtering scheme that can exploit the fact that all the heavy hitters are contained in $S_1 \times S_2$. (This is similar to special property of the estimation algorithms mentioned earlier.) For the base case of the recursion, we can use pretty much any filtering scheme (including the identity matrix). For the recursive steps, we use a filtering scheme using the ideas outlined in the previous paragraph but with a random code (which has excellent list recoverability) instead of the Parvaresh-Vardy type codes. As mentioned earlier this scheme cannot handle measurement noise. However, this scheme has another nice property: unlike the other construction, this one allows for a tradeoff between the decoding time and the number of measurement. Other than the claimed result, one can also obtain sublinear-time decoding (though not as efficient as poly(t) decoding time) while being within a constant factor of the optimal number of measurements.

Our result for general signals follows the list recoverability paradigm above but with Reed-Solomon codes instead of Parvaresh-Vardy codes. This leads to worse number of measurements but Reed-Solomon codes have better "distance" properties than Parvaresh-Vardy codes, which allows us to use the *same* matrix for both filtering and estimation. This allows us to have a strongly explicit construction, whereas the $O(d^{1+1/s} \log N (\log^{(m)} N)^s)$ result is randomized as the estimation step is randomized (while the filtering step is explicit). The estimation procedure is also extremely simple: just take the median of the measurements (of a filtered heavy hitter). We do not need the "pursuit" steps as in related prior works.

Even though list recoverable codes have been used to construct good group testing matrices [5, 13], to the best of our knowledge, this is the first work to explicitly use list recoverable codes in compressed sensing. Since sufficiently strong list recoverable codes are known to imply good expanders (cf. [12]), the work of [14] (and related papers) have used list recovery implicitly. However, the direct use of list recovery in our work leads to better parameters.

2 Coding Theory Facts

A code of *dimension* k and *block length* n over an *alphabet* Σ is a subset $C \subseteq \Sigma^n$ of *size* $|\Sigma|^k$. The *rate* of such a code equals k/n. Each vector in C is called a *codeword*. The *distance* of C is the minimum number of positions that any two distinct codewords differ in. A code with dimension k, block length n and distance Δ over Σ will be compactly referred to as an $(n, k, \Delta)_{|\Sigma|}$-code (or simply $(n, k)_{|\Sigma|}$-code if we do not care about its distance). A code C over $\mathbb{F}_q$ is called a *linear code* if C is a linear subspace of $\mathbb{F}_q^n$. A linear code with dimension k, block length n and distance Δ over $\mathbb{F}_q$ will be compactly referred to as an $[n, k, \Delta]_q$- (or simply $[n, k]_q$-) code.

A *concatenated binary code* has an *outer* code $C_{\text{out}} : [q]^{k_1} \to [q]^{n_1}$ over a large alphabet of size $q = 2^{k_2}$, and a binary *inner* code $C_{\text{in}} : \{0,1\}^{k_2} \to \{0,1\}^{n_2}$. The encoding of a message in $(\{0,1\}^{k_2})^{k_1}$ is natural. First, it is encoded with C_{out} and then C_{in} is applied to each of the outer codeword symbols. The concatenated code is denoted by $C_{\text{out}} \circ C_{\text{in}}$.

Let $\ell, L \geq 1$ be integers and let $0 \leq \alpha \leq 1$. A q-ary code C of block length n is called (α, ℓ, L)-*list recoverable* if for every sequence of subsets $S_1, \ldots, S_n \subseteq [q]$ such that $|S_i| \leq \ell$ for every $i \in [n]$, there exists at most L codewords $(c_1, \ldots, c_n)$ such that for at least αn positions i, $c_i \in S_i$. A $(1, \ell, L)$-list recoverable code will be henceforth referred to as (ℓ, L)-*zero error list recoverable*.

We will need the following powerful result due to Parvaresh and Vardy[2]:

▶ **Theorem 2.1** ([18]). *For all integers $s \geq 1$, for all prime powers r and all powers q of r, every pair of integers $1 < k \leq n \leq q$, there is an explicit $\mathbb{F}_r$-linear map $E : \mathbb{F}_q^k \to \mathbb{F}_{q^s}^n$ such that:*

1. *The image of E, $C \subseteq \mathbb{F}_{q^s}^n$, is a code of minimum distance at least $n - k + 1$.*
2. *Provided $\alpha > (s+1)(k/n)^{s/(s+1)}\ell^{1/(s+1)}$, C is an $(\alpha, \ell, O((rs)^s n\ell/k))$-list recoverable code. Further, a list recovery algorithm exists that runs in $\text{poly}((rs)^s, q, \ell)$ time.*

▶ **Corollary 2.2.** *Let $0 < \alpha \leq 1$ be a real and $s, \ell \geq 1$ be integers. Then for any prime power q, there is a strongly explicit $\mathbb{F}_p$ linear $\left(q, k \stackrel{def}{=} \frac{q}{s} \cdot \left(\frac{\alpha}{2s}\right)^{1+1/s} \cdot \frac{1}{\sqrt[s]{\ell}}\right)_{q^s}$ that is $\left(\alpha, \ell, s^{O(s)} q\ell/k\right)$-list recoverable in time $\text{poly}(ps^s, q, \ell)$.*

In the above, the s'th "order" Parvaresh-Vardy code will be referred to as the PV^s code. PV^1 is the well-known *Reed-Solomon codes* (RS code for short). For the RS codes, we will use the following result which has better list recoverability but has a much faster running time:

▶ **Theorem 2.3** ([2]). *An $[n,k]_q$ RS code is an $(\alpha, \ell, \sqrt{2n\ell/k})$-list recoverable provided $\alpha > \sqrt{\frac{2k\ell}{n}}$. Further, there is a $O(n^2\ell^2 \log(n\ell)\text{poly}(\log q))$-time list recovery algorithm.*

The next result is folklore.

▶ **Theorem 2.4.** *Let $0 < \alpha \leq 1$ be a real and let $\ell \geq 1$ be an integer. Then for integers $q \geq \frac{e^6\ell}{\alpha^2}$ and large enough n, the following holds. A random[3] $\left(n, k = \frac{\alpha n}{2 \log q}\right)_q$ code is $\left(\alpha, \ell, L \stackrel{def}{=} \lceil \frac{2\ell}{\alpha} \rceil\right)$-list recoverable with probability at least $1 - 2^{-\Omega(\alpha n L)}$.*

Concatenating PV codes with itself recursively m times leads to:

▶ **Corollary 2.5.** *Let $0 < \alpha \leq 1$ be a real and $s, m, \ell \geq 1$ be integers. Then the following holds for large enough n: there exists a $\left(n, \frac{n}{R}\right)_{q^s}$-code that is $(1-(1-\alpha)^m, \ell, s^{O(ms)}\ell/R)$-list recoverable where $R = \frac{1}{s} \cdot \left(\frac{\alpha}{2s}\right)^{1+1/s} \cdot \frac{1}{\sqrt[s]{\ell}}$, and $\frac{1}{R} \leq q \leq \frac{1}{R} \cdot \left(\log^{(m)} n + m\right)$.*

3 Constructions based on one-level code concatenation

We first fix some notations. For any $x \in \mathbb{R}^m$, $S \subseteq [m]$, let $x_S \in \mathbb{R}^m$ denote x with everything outside of S zeroed out. For any positive integer $d \leq m$, let $H_d(x)$ denote the set of d largest

[2] This statement of the theorem appears in [11].

[3] The q^k codewords in a random $(n,k)_q$ code are independently chosen by assigning each of the n symbols independently and uniformly at random from $[q]$.

(or "heaviest") coordinates of x in magnitudes, breaking ties arbitrarily. Note that, when $|S| \geq d$, we have $H_d(x_S) \subseteq S$.

3.1 General signals and Reed-Solomon-based compressed sensing

Our main result in this section is the following:

▶ **Theorem 3.1.** *For every $\epsilon > 0$, $M = M_{RS\circ \mathrm{ID}}$ with $c = O(1/\epsilon)$ is a compressed sensing measurement matrix with $t = O(d^2(\log_d N)^2/\epsilon^2)$ measurements which admits a $(d/\epsilon)^2\mathrm{poly}(\log N)$-time decoding algorithm that, given a noisy measurement vector $y = Mx+\nu$, outputs (the non-zero coordinates and values of) a d-sparse approximation $\hat{x} \in \mathbb{R}^N$ with the following approximation guarantee: $\|x - \hat{x}\|_1 \leq (1+\epsilon)\|x - x_{H_d(x)}\|_1 + \frac{0.4\epsilon}{\log N}\|\nu\|_1$.*

Fix a $[n, k, n-k+1]_q$-RS code $C = \{c_1, \cdots, c_{q^k}\} \subset [q]^n$, and the identity code $\mathrm{ID}_q : [q] \to \{0,1\}^q$. ($\mathrm{ID}_q(i)$ is the ith standard basis vector.) Let $M = M_{\mathrm{RS}\circ\mathrm{ID}}$ denote the matrix of the concatenated code $\mathrm{RS} \circ \mathrm{ID}$ defined as follows. The matrix M is a binary matrix with $N = q^k$ columns and $t = qn$ rows. We index each row of M by a pair $(p, i) \in [n] \times [q]$. Let M_j denote the jth column of M, then $M_j(p, i) = 1$ iff $c_j(p) = i$. We use M to compress signals in $\mathbb{R}^N$. Algorithm 1 is used for decoding. We will choose parameters n, k, q so that the RS code is $(\alpha = 1/2, l = cd, L = \sqrt{2nl/k})$-list recoverable. In particular, the parameters $q \geq n > k$ have to satisfy $1/2 > \sqrt{2kl/n}$, or $n > 8cdk$. Furthermore, we shall also choose $c > 2$.

Algorithm 1 Decoding algorithm for $M = M_{\mathrm{RS}\circ\mathrm{ID}}$

1: Input: $y \leftarrow Mx + \nu$, where ν is the noise vector, $c > 2$ and d is the sparsity parameter
2: // note again that $n > 8cdk$
3: **for** each position $p \in [n]$ **do**
4: Let $S_p \subset [q]$ denote the set of cd indices i such that the corresponding measurements $y(p, i)$ are the cd heaviest-magnitude measurements in the set $\{y(p, i) \mid i \in [q]\}$.
5: Use the list-recovery algorithm (with the inputs $S_p, p \in [n]$) for the RS code to recover a set $H \subset [N]$ of $\leq L$ indices.
6: **for** each $j \in H$ **do**
7: Let $\hat{x}_j = \mathrm{median}\{y(p, i) \mid c_j(p) = i\}$.
8: Return the top d (in magnitude) $\hat{x}_j, j \in H$.

For notational convenience, define $D = H_d(x) = H_d(x)$. Before presenting the two main steps of the analysis, we need a simple auxiliary lemma.

▶ **Lemma 3.2.** *Let $\delta = k/n$. Consider an arbitrary index $j \in [N]$. There is a subset $P \subset [n]$ of positions satisfying the following: (a) $|P| \geq 7(1-\delta d)n/8 \geq 3n/4$, and (b) for every $p \in P$, we have $|y(p, c_j(p)) - x_j| \leq \frac{10}{n}\|\nu\|_1 + \frac{8}{cd}\|x - x_D\|_1$.*

Proof. The RS code has relative distance $> 1-\delta$. Hence, every two codewords have at most δn positions with the same symbols. In particular, there is a set P' of at least $n - d\delta n = (1-\delta d)n$ positions satisfying the following: for every $p \in P'$, $c_j(p) \neq c_{j'}(p), \forall j' \in D \setminus \{j\}$.

Next, because for every $j' \in [N] \setminus (D \cup \{j\})$ the codeword $c_{j'}$ shares at most δn positions with c_j, the triangle inequality implies $\sum_{p\in P'} |y(p, c_j(p)) - x_j| \leq \|\nu\|_1 + \delta n\|x - x_D\|_1$. Since $|P'| \geq (1-\delta d)n$, by Markov inequality there is a subset $P \subset P'$ of at least $|P| \geq \frac{7}{8}(1-\delta d)n$ positions such that, for every $p \in P$

$$|y(p, c_j(p)) - x_j| \leq \frac{8(\|\nu\|_1 + \delta n\|x - x_D\|_1)}{(1-\delta d)n} \leq \frac{10}{n}\|\nu\|_1 + \frac{8}{cd}\|x - x_D\|_1.$$

The inequality $7(1-\delta d)n/8 \geq 3n/4$ follows from our assumptions that $n > 4ckd$ and $c > 2$. ◀

▶ **Lemma 3.3.** *Consider an arbitrary $j \in D$ such that $|x_j| > \frac{10\|x-x_D\|_1}{cd} + \frac{18\|\nu\|_1}{n}$. Then, $c_j(p) \in S_p$ for at least $\alpha n = n/2$ different positions $p \in [n]$ (where S_p is as defined in Step 4).*

Proof. Let $\delta = k/n$, and let P be the set of positions satisfying the properties stated in Lemma 3.2. Then, for each $p \in P$,

$$\begin{aligned} |y(p,c_j(p))| &\geq |x_j| - |y(p,c_j(p)) - x_j| > \frac{10\|x-x_D\|_1}{cd} + \frac{18\|\nu\|_1}{n} - \frac{10}{n}\|\nu\|_1 - \frac{8}{cd}\|x-x_D\|_1 \\ &\geq \frac{8}{cdn}\|\nu\|_1 + \frac{2}{cd}\|x-x_D\|_1. \end{aligned} \tag{1}$$

Next, for every position $p \in [n]$, let $\nu(p) = [\nu(p,1), \cdots, \nu(p,q)]$ denote the restriction of the noise vector ν on to the q coordinates within position p. Since $\|\nu\|_1 = \sum_{p=1}^{n} \|\nu(p)\|_1$, by Markov inequality there must be a set P' of at least $3n/4$ positions such that $\|\nu(p)\|_1 \leq 4\|\nu\|_1/n$ for every $p \in P'$. Let $\bar{P} = P \cap P'$. Then, $|\bar{P}| \geq n/2 = \alpha n$. We will prove that $c_j(p) \in S_p$ for every $p \in \bar{P}$, which would then complete the proof of the lemma.

Fix a position $p \in \bar{P}$. Let $y(p) = [y(p,1), \cdots, y(p,q)]$ be the subvector of y restricted to position p. For each $i \in [q]$, define $D_i = \{j \in D \mid c_j(p) = i\}$. By the triangle inequality and the fact that $p \in P'$, $\sum_{i \in S_p} |y(p,i) - \sum_{j \in D_i} x_j| \leq \|\nu(p)\|_1 + \|x - x_D\|_1 \leq \frac{4}{n}\|\nu\|_1 + \|x-x_D\|_1$. Noting that $|S_p| = cd$, by Markov inequality again there is a subset $S' \subset S_p$ of size $cd/2$ satisfying $|y(p,i) - \sum_{j \in D_i} x_j| \leq \frac{8}{cdn}\|\nu\|_1 + \frac{2}{cd}\|x-x_D\|_1$ for every $i \in S'$. Because $\sum_i |D_i| \leq d$ and $cd/2 > d$, there must be at least one $i' \in S'$ for which $D_{i'} = \emptyset$. Hence,

$$\min_{i \in S_p} |y(p,i)| \leq |y(p,i')| = |y(p,i') - \sum_{j \in D_i} x_j| \leq \frac{8}{cdn}\|\nu\|_1 + \frac{2}{cd}\|x-x_D\|_1. \tag{2}$$

From (1) and (2), and the fact that S_p contains the d largest coordinates in magnitudes of $y(p)$, we conclude that $i \in S_p$ as desired. ◀

We next prove that all the median estimates are pretty good.

▶ **Lemma 3.4.** *For any $j \in [N]$, $|x_j - \hat{x}_j| \leq \frac{10}{n}\|\nu\|_1 + \frac{8}{cd}\|x-x_D\|_1$.*

Proof. Let $\delta = k/n$, and let P be the set of positions satisfying the properties stated in Lemma 3.2. This means for $|P| \geq 3n/4 > n/2$ of the positions we know the values $y(p,c_j(p))$ are within $\pm\left(\frac{10}{n}\|\nu\|_1 + \frac{8}{cd}\|x-x_D\|_1\right)$ of x_j. Thus, so is the median $\hat{x}_j$ of the $y(p,c_j(p))$. ◀

We are now ready to prove the main result.

▶ **Theorem 3.5.** *With parameter $c = 18/\epsilon$, Algorithm 1 runs in time $\mathrm{poly}(d \log n)$ and outputs a d-sparse vector $\hat{x}$ satisfying $\|x-\hat{x}\|_1 \leq (1+\epsilon)\|x-x_D\|_1 + (28d/n)\|\nu\|_1$, where D is the set of k highest-magnitude coordinates of x.*

Proof. The total "extra mass" we get, relative to the best $\|x-x_D\|_1$, comes from the estimation error mass and the total mass of the small magnitude coordinates in D. It is not hard to see that

$$\begin{aligned} \|x-\hat{x}\|_1 &\leq \|x-x_D\|_1 + d\left(\frac{18\|\nu\|_1}{n} + \frac{10\|x-x_D\|_1}{cd}\right) + d\left(\frac{10}{n}\|\nu\|_1 + \frac{8}{cd}\|x-x_D\|_1\right) \\ &= (1+18/c)\|x-x_D\|_1 + (28d/n)\|\nu\|_1. \end{aligned}$$

◀

We next choose the parameters to minimize the number of measurements t of the matrix M. The following is not best possible (we can reduce it by a $\log\log$ factor). Choose $n = q$, $k = n/(4cd) = \epsilon n/(72d)$. For $N \leq q^k$, we need $\log N \leq \frac{\epsilon}{72d} q \log q$. Thus, we can pick $n = q = O\left(\frac{d}{\epsilon}\log_d N\right)$. The total number of measurements is $t = nq = O(\frac{d^2}{\epsilon^2}(\log_d N)^2)$.

Finally, we analyze the run time of the algorithm. The following steps dominate the running time of the other steps: (i) The first **for** loop, which takes $O(nq\log q) = O(\frac{d^2}{\epsilon^2}\text{poly}(\log_d N))$; (ii) Step 8 (the list recovery step), which by Theorem 2.3 takes $O(l^2n^2\log nl\text{poly}(\log q)) = d^2/\epsilon^2\text{poly}(\log N)$ time and (iii) the second **for** loop which takes $O(Lq) = O(d^2/\epsilon\log_d N)$ time. Thus, the overall running time is $d^2/\epsilon^2\text{poly}(\log N)$, as desired. This completes the proof of Theorem 3.1.

3.2 Non-negative signals

Our main result of this section is as follows:

▶ **Theorem 3.6.** *For every $\epsilon > 0$, and $s, m \geq 1$ there is a strongly explicit compressed sensing scheme with $t = (sm)^{O(s)} \cdot \left(\frac{d}{\epsilon}\right)^{1+1/s} \cdot (\log^{(m)} N + m)^s$ measurements and $\text{poly}(t)$ decoding time that on input a signal $x \in \mathbb{R}^N_{\geq 0}$ and measurement vector $\nu \in \mathbb{R}^t$ outputs a vector $\hat{x} \in \mathbb{R}^N$ that is d-sparse with the following approximation guarantee:*

$$\|x - \hat{x}\|_1 \leq (1+\epsilon)\|x - x_{H_d(x)}\|_1 + \frac{O(\epsilon)\cdot\log^{(m+1)} N}{\log N}\cdot\|\nu\|_1.$$

Estimation Algorithm. We will use the following estimation result by Porat and Strauss [19]. Our contribution is the filtering matrix.

▶ **Theorem 3.7** ([19]). *Let $N \geq d \geq 1$ be integers and $\epsilon > 0$ be a real. Then there exists a random $t \times N$ matrix M with the following properties: (i) $t = O\left(\frac{d}{\epsilon^2}\cdot\log(N/d)\right)$; and (ii) Let $S \subseteq [N]$, $x \in \mathbb{R}^N$ and $\nu \in \mathbb{R}^t$. Then there exists a $|S|\cdot(\log(N/d)/\epsilon)^{O(1)}$ time algorithm that given a noisy measurement $Mx + \nu$, outputs a vector $x' \in \mathbb{R}^N$ with at most $O(d)$ non-zero entries such that $\|x' - x_{H_d(x)}\|_1 \leq (1+\epsilon)\cdot(\|x - x_{H_d(x)}\|_1 + \|x_{H_d(x)\setminus S}\|_1) + \frac{c\cdot\epsilon}{\log(N/d)}\cdot\|\nu\|_1$, where $c \geq 1$ is some absolute constant.*

From list recovery to compressed sensing

▶ **Theorem 3.8.** *Let $d \geq 1$ be an integer and $c \geq 1$ be a real. Let $\ell, L \geq 1$ be integers. Then the following holds for any $q \geq \ell$, where q is a power of 2. Let C_{out} be an $(n_1, k_1)_q$-code that is $(1/2, \ell, L)$-list recoverable. Let C_{in} be an $(n_2, k_2 \stackrel{def}{=} \log q)_2$ code with the following property. For any vector $z \in \mathbb{R}^q_{\geq 0}$ and measurement noise $\mu \in \mathbb{R}^{n_2}$, given the measurement outcome $M_{C_{\text{in}}}z + \mu$, there is a $T_{in}(n_2, q)$-time algorithm that outputs at most ℓ coordinates of z containing the set $\left\{i \in [q] \mid z_i \geq \gamma\cdot\|z - z_{H_d(z)}\|_1 + \delta\cdot\|\mu\|_1\right\}$, where $\gamma, \delta > 0$. Then the matrix $M \stackrel{def}{=} M_{C_{\text{out}}\circ C_{\text{in}}}$ has the following properties:*

(i) *M is $t \times N$ matrix, where $t = n_1n_2$ and $N = q^{k_1}$.*

(ii) *For any $x \in \mathbb{R}^N_{\geq 0}$ and $\nu \in \mathbb{R}^t$, consider the noisy measurement vector $y = Mx + \nu \in \mathbb{R}^t_{\geq 0}$. There exists a set $H \subseteq [N]$ with $|H| \leq L$ such that*

$$\left\{i \in [N] \mid x_i \geq \gamma\cdot\|x_T\|_1 + 2\delta\cdot\frac{\|\nu\|_1}{n_1}\right\} \subseteq H, \tag{3}$$

where $T = [N]\setminus H_d(x)$.

(iii) *If C_{out} can be list recovered in time $T_{out}(n_1, q)$, then given y, the set H from part (ii) can be computed in time $n_1\cdot T_{in}(n_2, q) + T_{out}(n_1, q)$.*

Proof. Property (i) follows from the properties of concatenated codes. In the rest of the proof, we will prove (ii) and (iii) together by outlining an algorithm to compute the set H.

For notational convenience, let the codewords in C_{out} be denoted by $\{c_1, \ldots, c_N\}$. We will associate the ith coordinate of x with c_i.For any $j \in [n_1]$, let $\mu_j \in \mathbb{R}^{n_2}$ be the projection of ν to the positions in $[t]$ corresponding to the outer codeword position j. Note that $\|\nu\|_= \sum_{j\in[n_1]} \|\mu_j\|_1$. Thus by a Markov argument, there exists a subset $U \subseteq [n_1]$ with $|U| \geq n_1/2$ such that for every $j \in U$, $\|\mu_j\|_1 \leq \frac{2\|\nu\|_1}{n_1}$. Fix an outer codeword position $j \in U$. For any $i \in [N]$, let $c_i(j) \in [q]$ denote the jth symbol in the codeword c_i. Now define the vector $z = (z_1, \ldots, z_q)$ such that for any $\beta \in [q]$, $z_\beta = \sum_{i\in[N]:c_i(j)=\beta} x_i$. By the assumption in the Theorem statement and the upper bound on $\|\mu_j\|_1$, we know that in time $T_{in}(n_2, q)$ we can compute a set S_j of size at most ℓ such that $\left\{\beta \in [q] \mid z_\beta \geq \gamma \cdot \|z - z_{H_d(z)}\|_1 + 2\delta \cdot \frac{\|\nu\|_1}{n_1}\right\} \subseteq S_j$. Before we proceed we claim that

$$\|x_T\|_1 \geq \|z - z_{H_d(z)}\|_1. \tag{4}$$

Indeed, the above follows from the subsequent argument. Let $H' = \{c_i(j) | i \in H_d(x)\}$. Now note that $\|x_T\|_1 \geq \|z - z_{H'}\|_1 \geq \|z - z_{H_d(z)}\|_1$, where the first inequality follows from the definitions of z and H' while the second inequality follows from the definition of $H_d(z)$ and the fact that $|H'| \leq d$.

Thus, (4) (and the fact that x is a non-negative signal) implies that for every $i \in [N]$ such that $x_i \geq \gamma \cdot \|x_T\|_1 + 2\delta\|\nu\|_1/n_1$, $z_{c_i(j)} \geq x_i \geq \gamma \cdot \|z - z_{H_d(z)}\|_1 + 2\delta \cdot \|\nu\|_1/n_1$. In other words, $c_i(j) \in S_j$. As the choice of j was arbitrary, it holds that for every $i \in [N]$ such that $x_i \geq \gamma \cdot \|x_T\|_1 + 2\delta\|\nu\|_1/n_1$, $c_i(j) \in S_j$ for every $j \in U$. Recall that since $|S_j| \leq \ell$ for every $j \in U$, $|U| \geq n_1/2$ and as C_{out} is $(1/2, \ell, L)$-list recoverable in time $T_{out}(n_1, q)$ one can compute a set $H \subseteq [N]$ of size at most L such that every $i \in [N]$ such that $x_i \geq \gamma \cdot \|x_T\|_1 + 2\delta \cdot \|\nu\|_1/n_1$ satisfies $i \in H$. This completes the proof of (ii). Further, (iii) just follows from the description of the algorithm above to compute H. ◀

Specific Instantiations. The "identity" code $\text{ID}(q) : [q] \to \{0,1\}^q$ is often used as an inner code. Here, $\text{ID}(i)$ for any $i \in [q]$ is the q-bit vector that is set to 1 in position i and is zero otherwise. (The proofs are deferred to the full version.)

▶ **Lemma 3.9.** *Let $d \geq 1$ be an integer and $c \geq 1$ be a real. Let $q \geq 2(c+1)d$ be an integer. Then for any vector $x \in \mathbb{R}^q_{\geq 0}$ and measurement noise $\mu \in \mathbb{R}^q$, given the outcome $M_{\text{ID}(q)}x + \mu$, there is an $O(q \log(cd))$ time algorithm that outputs $\ell \stackrel{def}{=} 2(c+1)d$ coordinates of x such that it contains the set $T \stackrel{def}{=} \left\{i \in [q] \mid x_i \geq \frac{1}{cd} \cdot \|x - x_{H_d(x)}\|_1 + \frac{2}{(c+1)d} \cdot \|\mu\|_1\right\}$.*

Random CS construction. Applying Theorem 3.8 with the outer code from Theorem 2.4 as C_{out} (with q being a power of 2) and the code from Lemma 3.9 as C_{in} implies the following result (which we will use in Section 4):

▶ **Corollary 3.10.** *Let $N \geq d \geq 1$ be integers and $\epsilon > 0$ be a real. Then there exists a random $t \times N$ matrix M with the following properties: (i) $t = O\left(\frac{d}{\epsilon} \cdot \log N\right)$; and (ii) Let $S \subseteq [N]$, $x \in \mathbb{R}^N_{\geq 0}$ and $\nu \in \mathbb{R}^t$. Then there exists a $\tilde{O}(|S| \cdot t)$ time algorithm that given a noisy measurement $Mx + \nu$, outputs a subset $H \subseteq [N]$ with $|H| \leq O(d/\epsilon)$ such that $S \cap \left\{i \in [N] \mid x_i \geq \frac{\epsilon}{d} \cdot \|x - x_{H_d(x)}\|_1 + \frac{\epsilon}{d \cdot \log N} \cdot \|\nu\|_1\right\} \subseteq H$.*

Explicit CS construction. Applying Theorem 3.8 with the outer code from Corollary 2.5 (with q being a power of 2) and the inner code from Lemma 3.9 implies the following result:

▶ **Corollary 3.11.** *Let $n \geq d \geq 1$ and $s, m \geq 1$ be integers and $\epsilon > 0$ be reals. Then there exists an explicit $t \times N$ matrix M with the following properties:*

- $t \leq (sm)^{O(s)} \cdot \left(\frac{d}{\epsilon}\right)^{1+1/s} \cdot (\log^{(m)} N + m)^s$
- *Let $x \in \mathbb{R}^N_{\geq 0}$ and $\nu \in \mathbb{R}^t$. Then there exists a $\mathrm{poly}(t)$ time algorithm that given a noisy measurement $Mx + \nu$, outputs a subset $H \subseteq [N]$ with $|H| \leq$ such that $H \supseteq \left\{i \in [N] | x_i \geq \frac{\epsilon}{d} \cdot \|x - x_{H_d(x)}\|_1 + \frac{\epsilon \cdot \log^{(m+1)} N}{d \cdot \log N} \cdot \|\nu\|_1\right\}$.*

The above (instantiated with $C_{\text{out}} = \text{PV}^s$ and $C_{\text{in}} = \text{ID}$) with Theorem 3.7 proves Theorem 3.6.

4 Recursive construction with multiple recursion levels

Our main result of this section is the following:

▶ **Theorem 4.1.** *For every $\epsilon > 0$, there is a randomized compressed sensing scheme with $t = O(d/\epsilon^2 \log N \log \log_d N)$ measurements and $\mathrm{poly}(t)$ decoding time that on input a signal $x \in \mathbb{R}^N_{\geq 0}$ outputs a vector $\hat{x} \in \mathbb{R}^N$ that is d-sparse with the following approximation guarantee:*

$$\|x - \hat{x}\|_1 \leq (1 + \epsilon)\|x - x_{H_d(x)}\|_1.$$

The main technical result to prove the above theorem is the following result:

▶ **Theorem 4.2.** *Let $n \geq d \geq 1$ be integers. Assume for every $i \geq d$, there is a $t(i) \times i$ matrix $\mathbf{M}_i$ with the following property. For any subset $S \subseteq [i]$, vector $z \in \mathbb{R}^i_{\geq 0}$ and given the outcome vector $\mathbf{M}_i z$, there is a $d(|S|, i)$-time algorithm that outputs at most ℓ coordinates of z containing the set*

$$\{i \in S \mid z_i \geq \gamma \cdot \|z - z_{H_d(z)}\|_1\}, \tag{5}$$

where $\gamma > 0$. Let $1 \leq a \leq \log n$ and $1 \leq b \leq \log n/a$ be integers. Then there exists a $t_{a,b} \times n$ matrix $\mathbf{M}_{a,b}$ that has the following property. Given any $x \in \mathbb{R}^n_{\geq 0}$, from the measurement vector $\mathbf{M}_{a,b}x$, in time $D_{a,b}$, one can compute a set H with $|H| \leq \ell$ such that

$$H \supseteq \{i \in [n] \mid x_i \geq \gamma \cdot \|x - x_{H_d(x)}\|_1\}, \tag{6}$$

where

$$t_{a,b} = \sum_{j=0}^{\lceil \log_b(\frac{\log n}{a}) \rceil - 1} b^j \cdot t\left(\sqrt[b^j]{n}\right) \tag{7}$$

and

$$D_{a,b} = \sum_{j=0}^{\lceil \log_b(\frac{\log n}{a}) \rceil - 2} b^j \cdot d\left(\ell^b, \sqrt[b^j]{n}\right) + \frac{\log n}{a} \cdot d(2^a, 2^a). \tag{8}$$

Finally, if the family of matrices $\{\mathbf{M}_i\}_{i \geq d}$ is (strongly) explicit then so is $\mathbf{M}_{a,b}$.

Proof. We will construct the final matrix $\mathbf{M}_{a,b}$ recursively. In particular, let such a matrix in the recursion with N columns be denoted by $\mathbf{M}_{a,b}(N)$. Note that the final matrix is $\mathbf{M}_{a,b} = \mathbf{M}_{a,b}(n)$. (For notational convenience, we will define $D_{a,b}(N)$ and $t_{a,b}(N)$ to be the decoding time for and the number of rows in $\mathbf{M}_{a,b}(N)$respectively). Next, we define the recursion. If $N \leq 2^a$, then set $\mathbf{M}_{a,b}(N) = \mathbf{M}_N$. Note that in this case, $t_{a,b}(N) = t(N)$. Further, we will

use the given algorithm in the base case, which implies that $D_{a,b}(N) = d(N, N)$. It is easy to check that both (7) and (8) are satisfied. Finally since $\mathbf{M}_N$ satisfies (5), $\mathbf{M}_{a,b}(N)$ satisfies (6).

Now consider the case when $N > 2^a$. For $i \in [b]$, define $\mathbf{M}^{(i)}$ to be the $t_{a,b}(\sqrt[b]{N}) \times N$ matrix whose kth column (for $k \in [N]$) is identical to the mth column in $\mathbf{M}_{a,b}(\sqrt[b]{N})$ where m is the ith chunk of $\frac{1}{b}\log n$ bits in k (e think of k and m as their respective binary representations). Define $\mathbf{M}_{a,b}(N)$ to be the stacking of $\mathbf{M}^{(1)}, \mathbf{M}^{(2)}, \ldots, \mathbf{M}^{(b)}$ and $\mathbf{M}_N$. First, we verify that (7) holds. To this end note that

$$t_{a,b}(N) = b \cdot t_{a,b}(\sqrt[b]{N}) + t(N). \tag{9}$$

In particular, (by induction) all the $\mathbf{M}^{(i)}$ contribute $b \cdot \sum_{j=0}^{\left\lceil \log_b\left(\frac{\log \sqrt[b]{N}}{a}\right)\right\rceil - 1} b^j \cdot t\left(\sqrt[b^j]{\sqrt[b]{N}}\right) = \sum_{j=1}^{\lceil \log_b(\frac{\log N}{a})\rceil - 1} b^j \cdot t\left(\sqrt[b]{N}\right)$ rows. Since $\mathbf{M}_N$ adds another $t(N)$ rows, $\mathbf{M}_{a,b}(N)$ indeed satisfies (7).

Finally, we consider the decoding of $\mathbf{M}_{a,b}(N)$. The decoding algorithm is natural: we run the decoding algorithm for $\mathbf{M}_{a,b}(\sqrt[b]{N})$ (that is guaranteed by induction) on the part of the outcome vector corresponding to each of the $\mathbf{M}^{(i)}$ ($i \in [b]$) to compute sets S_i with the following guarantee.

Let $z^{(i)}$ (for $i \in [b]$) be defined as follows. For any $0 \le j \le \sqrt[b]{N} - 1$, the jth entry of $z^{(i)}$ is the sum of the x_k's where the ith chunk of $\log n/b$ bits in k is the same as j (where we think of k and j as $\log n$-bit and $\log n/b$-bit vectors respectively). By induction, when the decoding algorithm for $\mathbf{M}_{a,b}(\sqrt[b]{N})$ is run on $\mathbf{M}^{(i)}z^{(i)}$, then it outputs a set S_i with $|S_i| \le \ell$ such that $\{j \in [\sqrt[b]{N}] \mid z^{(i)}_j \ge \gamma \cdot \|z^{(i)} - z^{(i)}_{H_d(z^{(i)})}\|_1\} \subseteq S_i$. Finally, we run the algorithm for $\mathbf{M}_N$ on $\mathbf{M}_N x$ given the set $S \stackrel{def}{=} S_1 \times S_2 \times \cdots \times S_b$ to obtain a set H with $|H| \le \ell$. To show that H satisfies (6), we need to show that $\{j \in [N] \mid x_j \ge \gamma \cdot \|x - x_{H_d(x)}\|_1\} \subseteq S$. In particular, we need to show that for any $j \in [N]$ with $x_j \ge \gamma \cdot \|x - x_{H_d(x)}\|_1$, $j_i \in S_i$ for every $i \in [\sqrt[b]{N}]$, where j_i denotes the ith chunk of $\log n/b$ bits in j (where we think of j as a $\log N$-bit vector). To this end, we first note that using the same argument as in Theorem 3.8, we have $\|x - x_{H_d(x)}\|_1 \ge \|z^{(i)} - z^{(i)}_{H_d(z^{(i)})}\|_1$. Since x is a non-negative signal (and the definition of $z^{(i)}$), it is easy to see that if $x_j \ge \gamma \cdot \|x - x_{H_d(x)}\|_1$, then $z^{(i)}_{j_i} \ge \gamma \cdot \|z^{(i)} - z^{(i)}_{H_d(z^{(i)})}\|_1$, which in turn implies $j_i \in S_i$, as desired.

To complete the proof, we need to verify that this algorithm takes time as claimed in (8). Note that $D_{a,b}(N) = b \cdot D_{a,b}(\sqrt[b]{N}) + t(\ell^b, N)$. The rest of the proof of (8) is similar to that of (7). Finally, the claim on explicitness follows from the construction. ◀

Applying Theorem 4.2 to Corollary 3.10, we obtain

▶ **Corollary 4.3.** *Let $N \ge d \ge 1$ be integers and $\epsilon > 0$ be a real. Then there exists a random $t \times N$ matrix M with the following properties: (i) $t = O\left(\frac{d}{\epsilon} \cdot \log N \log\log_d N\right)$; and (ii) Let $x \in \mathbb{R}^N_{\ge 0}$. Then there exists a* poly(t) *time algorithm that given a measurement Mx, outputs a subset $H \subseteq [N]$ with $|H| \le O(d/\epsilon)$ such that $H \supseteq \left\{i \in [N] | x_i \ge \frac{\epsilon}{d} \cdot \|x - x_{H_d(x)}\|_1\right\}$.*

The above (with the $\mathbf{M}_i$ family following from the construction in Theorem 3.8 instantiated with a random code as C_{out} and the identity code as the inner code) with Theorem 3.7 implies Theorem 4.1.

References

1 Rudolf Ahlswede and Vladimir B. Balakirsky. Construction of uniquely decodable codes for the two-user binary adder channel. *IEEE Trans. Inform. Theory*, 45(1):326–330, 1999.

2 Daniel Augot and Lancelot Pecquet. A hensel lifting to replace factorization in list-decoding of algebraic-geometric and reed-solomon codes. *IEEE Transactions on Information Theory*, 46(7):2605–2614, 2000.

3 Khanh Do Ba, Piotr Indyk, Eric Price, and David P. Woodruff. Lower bounds for sparse recovery. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1190–1197, 2010.

4 Emmanuel J. Candès, Justin K. Romberg, and Terence Tao. Robust uncertainty principles: exact signal reconstruction from highly incomplete frequency information. *IEEE Transactions on Information Theory*, 52(2):489–509, 2006.

5 Mahdi Cheraghchi. Noise-resilient group testing: Limitations and constructions. In *Proceedings of the 17th International Symposium on Fundamentals of Computation Theory (FCT'2009)*, pages 62–73, 2009.

6 Graham Cormode and S. Muthukrishnan. Combinatorial algorithms for compressed sensing. In *13th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 280–294, 2006.

7 David L. Donoho. Compressed sensing. *IEEE Transactions on Information Theory*, 52(4):1289–1306, 2006.

8 Anna C. Gilbert and Piotr Indyk. Sparse recovery using sparse matrices. *Proceedings of the IEEE*, 98(6):937–947, 2010.

9 Anna C. Gilbert, Martin J. Strauss, Joel A. Tropp, and Roman Vershynin. One sketch for all: fast algorithms for compressed sensing. In *Proceedings of the 39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 237–246, 2007.

10 Venkatesan Guruswami. *List decoding of error-correcting codes*. Number 3282 in Lecture Notes in Computer Science. Springer, 2004.

11 Venkatesan Guruswami and Atri Rudra. Soft decoding, dual BCH codes, and better list-decodable ϵ-biased codes. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity (CCC)*, pages 163–174, 2008.

12 Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from parvaresh-vardy codes. In *Proceedings of the 22nd Annual IEEE Conference on Computational Complexity*, pages 96–108, 2007.

13 Piotr Indyk, Hung Q. Ngo, and Atri Rudra. Efficiently decodable non-adaptive group testing. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1126–1142, 2010.

14 Piotr Indyk and Milan Ruzic. Near-optimal sparse recovery in the l1 norm. In *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 199–207, 2008.

15 W. H. Kautz and R. C. Singleton. Nonrandom binary superimposed codes. *IEEE Trans. Inf. Theory*, 10:363–377, 1964.

16 S. Muthukrishnan. *Data Streams: Algorithms and Applications*, volume 1. NOW, 2005.

17 Hung Q. Ngo, Ely Porat, and Atri Rudra. Efficiently decodable error-correcting list disjunct matrices and applications. In *ICALP (1)*, pages 557–568, 2011.

18 Farzad Parvaresh and Alexander Vardy. Correcting errors beyond the Guruswami-Sudan radius in polynomial time. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 285–294, 2005.

19 Ely Porat and Martin Strauss. Sublinear time, measurement-optimal, sparse recovery for all, July 2012. To be presented at SODA 2012. Also arXiv:1012.1886 [cs.DS].

Ehrenfeucht-Fraïssé goes elementarily automatic for structures of bounded degree

Antoine Durand-Gasselin and Peter Habermehl

Univ Paris Diderot, Sorbonne Paris Cité, LIAFA, UMR 7089 CNRS, F-75205 Paris, France
{adg,haberm}@liafa.jussieu.fr

Abstract

Many relational structures are automatically presentable, i.e. elements of the domain can be seen as words over a finite alphabet and equality and other atomic relations are represented with finite automata. The first-order theories over such structures are known to be primitive recursive, which is shown by the inductive construction of an automaton representing any relation definable in the first-order logic. We propose a general method based on Ehrenfeucht-Fraïssé games to give upper bounds on the size of these automata *and* on the time required to build them. We apply this method for two different automatic structures which have elementary decision procedures, Presburger Arithmetic and automatic structures of bounded degree. For the latter no upper bound on the size of the automata was known. We conclude that the very general and simple automata-based algorithm works well to decide the first-order theories over these structures.

1998 ACM Subject Classification F.4.1 Computational Logic; F.2.2 Computations on discrete structures

Keywords and phrases Automata-based decision procedures for logical theories, Automatic Structures, Ehrenfeucht-Fraïssé Games, Logics, Complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.242

1 Introduction

The idea of automatic structure first appeared in the work of Büchi and Elgot [1, 4] who showed how to use finite word automata to decide the weak second-order theory of integers with one successor and hence Presburger Arithmetic. Hodgson [7] exhibited that a general effective procedure to build an automaton whose language corresponds exactly to the solutions of a first-order formula over a relational structure can be given, if the basic relations can be described by automata. Khoussainov and Nerode [8] called these structures automatic structures and initiated a systematic study of which structures can be automatically presented.

The construction of an automaton accepting the solutions of a first order formula for an automatic structure is very simple. It can be done inductively on the structure of the formula by replacing the logical operators by corresponding operations on (deterministic) automata. For example, existential quantification can be done by projection and determinisation. The complexity in general is known to be primitive recursive, which is a tight bound since some automatic structures have a non-elementary first-order theory. Some work on the size of these automata has been done by Klaedtke [10] and Eisinger [3] for some (ω-)automatic structures, and for the well-studied Presburger Arithmetic an optimal time upper bound for the size [9] of the automaton and for its construction [2] has been obtained. In [3, 10] Ehrenfeucht-Fraïssé games are used as a proof tool. These games have been classically used to show bounds for the decision procedure of logical theories by using the fact that quantification over an infinite set can be replaced by quantification over some finite set (see e.g. [5]). First Klaedtke [10]

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 242–253

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


and then Eisinger [3] linked this approach with the automata approach by relating the states of a minimal automaton corresponding to a formula with equivalence classes (whose number can be bounded) determined by a suitable Ehrenfeucht-Fraïssé game. In [2] we obtain an upper bound for the complexity of the construction of the automata in a different way.

For automatic structures of bounded degree (i.e. elements of the domain are only in relation with a bounded number of other elements), several elementary complexity results were shown recently [11], notably a 2EXPSPACE algorithm for the uniform model-checking problem of injective automatic structures. These results are shown also via a kind of Ehrenfeucht-Fraïssé argument using Gaifman's locality principle [6] but the decision procedure is neither based on the inductive automata construction nor easily practically implementable. As far as we know, no upper bound on the size and the construction of the automaton corresponding to solutions of a formula has been shown. One result of this paper is a 3EXPTIME upper bound for this problem, using the simple inductive automaton construction.

To obtain this result we present an extension of Klaedtke's approach of the use of Ehrenfeucht-Fraïssé games to automatic structures. Roughly speaking, Klaedtke's approach consists in relating states of a minimal automaton of a formula to equivalence classes of suitably chosen refinements of relations defined by Ehrenfeucht-Fraïssé games (called Ehrenfeucht-Fraïssé relations). Showing an upper bound on the index of these relations then gives an upper bound on the size of the minimal automaton for a given formula. We use the same kind of relations and give in our main theorem general conditions allowing to obtain an upper bound even on the time needed to construct the automaton. Even though the automata constructed are in general not minimal, we show that they satisfy the crucial property that two words in the same equivalence class of the Ehrenfeucht-Fraïssé relation must lead to the same state of an automaton (even after determinisation of an automaton obtained by projection). This allows to obtain a bound on the size of all automata inductively constructed depending on the index of the Ehrenfeucht-Fraïssé relation.

We also apply our main theorem to Presburger Arithmetic (with *most*-significant digit first encoding), using very similar Ehrenfeucht-Fraïssé relations as [3] who shows a triple exponential upper bound on the size of the automaton, and we extend his results to show that even the construction can be done in 3EXPTIME. The same result for *least*-significant digit first encoding was shown in [2] using a complicated analysis of the automata obtained from quantifier-free formulas like in [9].

The paper is organised as follows. We first recall the notion of automatic structures and an explicit inductive construction of an automaton that accepts solutions of a formula. Then we present our main theorem which gives conditions on Ehrenfeucht-Fraïssé relations that imply upper bounds on the size (and the time required to inductively build it) of the automaton accepting solutions of a first-order formula. Finally, we show how to define Ehrenfeucht-Fraïssé relations allowing to apply our main theorem for automatic structures of bounded degree and for Presburger Arithmetic.

2 Preliminaries

We suppose that the reader is familiar with finite string (over finite alphabet) automata. We define the size of an automaton $A = (\Sigma, Q, q_0, F, \delta)$ as the space required to write it. We will only consider automata with alphabets such that letters are written in space logarithmic w.r.t. the size of the alphabet, and whose states are integers (ranging from 1 to the number of states). It is clear that the size of such an automaton is bounded by some polynomial in $|\Sigma|$ and $|Q|$. Many manipulations over deterministic automata (complementation, minimisation,

product of two automata, relabelling) can be performed within time polynomial w.r.t. the size of the input automata. In the following we use on-the-fly constructions, i.e. states in the constructed automata are created on demand; only reachable states are considered. Although the time (and space) complexity of automaton determinisation (using the well-known subset construction) can't be bounded by any polynomial of the size of the input, it is clear that it can be bounded by some fixed polynomial of the sizes of the input *and* the (trim) output automata. In the following we use mainly the same notation as [11].

2.1 Structures and first-order logic

A *signature* is a finite set $\mathcal{S}$ of *predicate* symbols. Each predicate symbol $P \in \mathcal{S}$ has a fixed arity denoted by ar_P. A *relational structure* of signature $\mathcal{S}$ is a couple $\mathcal{A} = (A, (P^{\mathcal{A}})_{P\in\mathcal{S}})$, with A a set called *domain* and $P^{\mathcal{A}} \subseteq A^{ar_P}$. We will identify a predicate P with its *interpretation* $P^{\mathcal{A}}$. We say that P holds for $(a_1, \ldots, a_{ar_P}) \in A^{ar_P}$ (also formulated $P(a_1, \ldots, a_{ar_P})$ holds) if $(a_1, \ldots, a_{ar_P}) \in P^{\mathcal{A}}$. A *congruence* on the relational structure $\mathcal{A} = (A, (P^{\mathcal{A}})_{P\in\mathcal{S}})$ is an equivalence relation $\equiv$ on A such that for all $P \in \mathcal{S}$ and $a_1, \ldots, a_{ar_P}, b_1, \ldots, b_{ar_P} \in A$ such that $a_i \equiv b_i$ (for any $i \leq ar_P$), we have that if $P(a_1, \ldots, a_{ar_P})$ holds, then $P(b_1, \ldots, b_{ar_P})$ holds as well. We denote by $[a]_{\equiv}$ (or $[a]$) the equivalence class of $a \in A$ w.r.t. $\equiv$. $A/_{\equiv}$ denotes the set of all equivalence classes. For each predicate P we can define the quotient predicate $P/_{\equiv}$ by $P/_{\equiv}([a_1], \ldots, [a_{ar_P}])$ holds iff $P(a_1, \ldots, a_{ar_P})$ holds. Furthermore the quotient structure $\mathcal{A}/_{\equiv}$ is defined as the structure $(A/_{\equiv}, (P/_{\equiv})_{P\in\mathcal{S}})$.

We write $\bar{\alpha}_r$ as a shorthand for $(\alpha_1, \ldots, \alpha_r)$, with the α_i possibly being elements of a set (typically A), or variables. We also write $[1, r]$ to denote the set of integers between 1 and r.

First-order formulas over the signature $\mathcal{S}$ are defined as usual as either:

- an atomic formula $\varphi(\bar{x}_r)$ over r variables ($\bar{x}_r$), i.e. of the form $P(x_{j_1}, \ldots, x_{j_{ar_P}})$ for some predicate P with arity ar_P and $(j_k)_{1\leq k\leq ar_P}$ some ar_P-tuple of elements in $[1, r]$. Notice that some variables may not appear syntactically in the formula whereas others may appear more than once. The size of such a formula is defined as $\|\varphi\| = ar_P$.
- a conjunction, $\varphi(\bar{x}_r) = \varphi_1(\bar{x}_r) \wedge \varphi_2(\bar{x}_r)$ with φ_1 and φ_2 two first order formulas over the *same*[1] r variables. We define its size to be $\|\varphi\| = 1 + \|\varphi_1\| + \|\varphi_2\|$.
- a negation, $\varphi(\bar{x}_r) = \neg\varphi_1(\bar{x}_r)$ with φ_1 a formula over r variables; $\|\varphi\| = 1 + \|\varphi_1\|$.
- or an existential quantification, i.e. $\varphi(\bar{x}_r) = \exists y.\varphi_1(\bar{x}_r, y)$ where y is a fresh variable and φ_1 a formula over $r + 1$ variables; $\|\varphi\| = 1 + \|\varphi_1\|$.

Given a formula $\varphi(\bar{x}_r)$ over r variables, we denote by $\mathcal{A} \vDash \varphi(\bar{a}_r)$ (with $\bar{a}_r \in A^r$) that the formula φ is valid (in the usual sense) when we substitute the variables with the corresponding constants. We can associate to any formula its set of solutions (in the structure $\mathcal{A}$ which will always be clear from the context), that is the set of assignments of the free variables seen as r-tuples of elements of A that satisfy (in the usual sense) the formula. Thus first-order (r-variables) formulas define (r-ary) *first-order relations* over the domain.

2.2 Automatic presentations

Informally, an automatic structure is a relational structure whose domain can be represented by a regular language over an alphabet Σ such that ar-ary predicates can also be seen

[1] This will be useful for defining easily the automata corresponding to a formula. Notice that this is not a restriction, since if two formulas do not syntactically contain the same variables, we can consider that they have the same variables by adding them implicitly, e.g. in $P(x_1, x_2) \wedge P(x_2, x_3)$ both subformulas have free variables x_1, x_2, x_3.

as regular languages. We extend the representation of the domain as a regular language to a representation of any cartesian power of the domain as a regular language. As we can't represent a k-tuple of words of Σ^* with a word over the alphabet Σ^k —the k words do not necessarily have the same length— we pad the shorter words with an additional symbol $\diamond \notin \Sigma$. We repeatedly add $\diamond$ at the beginning of the shorter words (rather than at the end as done usually) giving the k words the same length leading to the definition of (right-aligning) *convolution* of words: Let $\bar{a}_k$ be a k-tuple of words in Σ^*. We write $\langle \bar{a}_k \rangle$ for its convolution which is a word over the alphabet $(\Sigma \cup \{\diamond\})^k \backslash \{\diamond\}^k$ (denoted by $\hat{\Sigma}_k$). Its length is $|\langle \bar{a}_k \rangle| = \max_i(|a_i|)$ where $|a_i|$ denotes the length of a_i and its j-th letter (starting from 1) is $(a_1[|a_1| - |\langle \bar{a}_k \rangle| + j], a_2[|a_2| - |\bar{a}_k| + j], \ldots, a_k[|a_k| - |\langle \bar{a}_k \rangle| + j])$ where $a_i[|a_i| - |\langle \bar{a}_k \rangle| + j]$ denotes $\diamond$, if $(|a_i| - |\langle \bar{a}_k \rangle| + j)$ is not strictly positive, and the $(|a_i| - |\langle \bar{a}_k \rangle| + j)$-th letter of a_i otherwise. For example, we have $\langle bab, \epsilon, bc \rangle = (b, \diamond, \diamond)(a, \diamond, b)(b, \diamond, c)$. Conversely, we define the operators $(.\downarrow_i)_{1 \leq i \leq k}$ and $(.\Downarrow_i)_{1 \leq i \leq k}$ for words in $\hat{\Sigma}_k^*$. $.\downarrow_i$ is a monoïd morphism from $\hat{\Sigma}_r^*$ to $(\Sigma \cup \{\diamond\})^*$ projecting each letter of the word to its i-th component. $w\Downarrow_i$ is defined as the greatest suffix of $w\downarrow_i$ not starting with $\diamond$. We write $\overline{w\Downarrow}_k$ as a shorthand for $(w\Downarrow_1, \ldots, w\Downarrow_k)$. The duality between convolution and $\Downarrow_.$ is exhibited by the identity: $\bar{a}_k = \overline{\langle \bar{a}_k \rangle \Downarrow}_k$.

▶ **Definition 1.** An *r-variable automaton* A over Σ is a finite automaton over the alphabet $\hat{\Sigma}_r$ such that $L(A) \subseteq \{\langle \overline{w}_r \rangle \mid \forall i \in [1, r], w_i \in \Sigma^*\}$. It represents the r-ary relation $R(A) = \{\overline{w\Downarrow}_k \mid w \in L(A)\}$.

Let us notice that provided letters of the alphabet Σ can be written within space logarithmic w.r.t. $|\Sigma|$, letters of $\hat{\Sigma}_r$ can be written within space logarithmic w.r.t. $|\hat{\Sigma}_r|$. Thus most operations over r-variable automata will also be achieved within polynomial time.

▶ **Definition 2.** An *automatic presentation* is a tuple $AP = (\Sigma, \mathcal{S}, A_D, A_=, (A_P)_{P \in \mathcal{S}})$ where Σ is a finite alphabet, $\mathcal{S}$ is a signature, A_D is an automaton over Σ, $(A_P)_{P \in \mathcal{S}}$ is a family of ar_P-variable automata over Σ and $A_=$ is a 2-variable automaton over Σ such that $R(A_=)$ is a congruence on the structure $(L(A_D), (R(A_P))_{P \in \mathcal{S}})$.

An automatic presentation AP is called deterministic, if all its automata are deterministic. Its size $\|AP\|$ is the space required to write all its automata. The structure *presented* by AP is the quotient $\mathcal{A}(AP) = (L(A_D), (R(A_P))_{P \in \mathcal{S}})/_{R(A_=)}$. AP is *injective* if $R(A_=)$ is the identity relation. A relational structure is called *automatically presentable* (or automatic) if there is an automatic presentation isomorphic to it. The element $[w]_{R(A_=)}$ with $w \in L(A_D)$ of the structure $\mathcal{A}(AP)$ is denoted by $[w]$. Given $u \in \hat{\Sigma}_r^*$ a convolution of r words in Σ^* we say that u represents $([u\Downarrow_1], \ldots, [u\Downarrow_r])$.

2.3 Automata-based model-checking

We are interested in the following problem.

▶ **Definition 3.** The model-checking problem for a relational structure $\mathcal{A} = (A, (P)_{P \in \mathcal{S}})$ over a signature $\mathcal{S}$ and a first-order sentence φ over the same $\mathcal{S}$ is to decide whether $\mathcal{A} \vDash \varphi$.

For automatic structures, this problem has been shown decidable using the following theorem [7, 8]. It provides also a way to get a representation of all solutions of a formula.

▶ **Theorem 4.** *Given an automatic presentation $AP = (\Sigma, \mathcal{S}, A_D, A_=, (A_P)_{P \in \mathcal{S}})$ and a first-order formula φ over $\mathcal{S}$ with r free variables one can build an r-variable automaton A_φ over Σ such that $R(A_\varphi) = \{(w_1, \ldots, w_r) \in L(A_D)^r \mid \mathcal{A}(AP) \vDash \varphi([w_1], \ldots, [w_r])\}$.*

In Section 3 we study the complexity of the automaton construction, i.e. the size of the automaton A_φ corresponding to a formula φ as well as the time needed to construct it. This in turn gives complexity bounds for the model-checking problem. We first give here a detailed description of the automaton construction. We consider *deterministic* automatic presentations. Given a formula φ (with r free variables) we have to build an automaton that distinguishes vectors of elements of the domain whose representatives satisfy the formula from those that don't. Intuitively, it is straightforward to build such automata inductively. We first give the construction of the r-variable minimal automaton A_{D^r} that accepts exactly all convolutions of words in $L(A_D)$. We will ensure we only build automata which reject any word not representing a convolution of words in $L(A_D)$ by product with A_{D^r}.

To construct A_{D^r} we build an automaton accepting $\diamond^* L(A_D)$ denoted by $A'_D = (\Sigma \cup \{\diamond\}, Q'_D, q_0, F, \delta)$ which has just one more state than A_D. Then we construct $A'_{D^r} = (\hat{\Sigma}_r, Q_{D^r}, q'_0, F', \delta_r)$ on-the-fly as follows: Q_{D^r} is the subset of $(Q'_D)^r$ reached by the on-the-fly construction, $q'_0 = (q_0, \ldots, q_0)$, $F' = F^r$ and δ_r is defined as: let $q'_i = \delta(q_i, a_i)$ for all i, then $\delta_r((q_1, \ldots, q_r), \bar{a}_r) = (q'_1, \ldots, q'_r)$. Finally, A_{D^r} is obtained by minimising A'_{D^r}. It is clear that the time to build this minimal automaton is bounded by some polynomial of $\|AP\|^r$: indeed as $\|AP\|$ is greater than both $|\Sigma|$ and the number of states in A_D, $\|AP\|^r$ is greater than the number of states of A'_{D^r} and the size of its alphabet. Remark that the fact that A_{D^r} is minimal is needed later in Section 3.

We now detail an inductive (on the structure of the formula) construction of the r-variable automaton accepting representatives of solutions of some formula φ with r free variables.

Let's start by the **case of atomic formulas**, i.e. of the form $\varphi(\bar{x}_r) = P(x_{j_1}, \ldots, x_{j_{ar}})$ with P a predicate of $\mathcal{S}$ with arity ar, and the $(j_k)_{1 \leq k \leq ar}$ a tuple of ar integers in $[1, r]$. The construction of the r-variable automaton $A_{P(x_{j_1}, \ldots, x_{j_{ar}})}$ is performed in two steps: first we build $A'_{P(x_{j_1}, \ldots, x_{j_{ar}})}$ which within words corresponding to a convolution of words in $L(A_D)$ accepts only those satisfying $P(x_{j_1}, \ldots, x_{j_{ar}})$. As we introduce extra tracks for variables not appearing in $P(x_{j_1}, \ldots, x_{j_{ar}})$, this automaton may accept words that are not convolutions of words in $L(A_D)$. Therefore we build $A_{P(x_{j_1}, \ldots, x_{j_{ar}})}$ as the minimal automaton accepting the intersection of languages of $A'_{P(x_{j_1}, \ldots, x_{j_{ar}})}$ and A_{D^r}. Let $A_P = (\hat{\Sigma}_{ar}, Q_P, q_0, F_P, \delta_P)$, then $A'_{P(x_{j_1}, \ldots, x_{j_{ar}})} = (\hat{\Sigma}_r, Q_P, q_0, F_P, \delta')$ where δ' is given as follows: for all $q \in Q_P$ and $(l_1, \ldots, l_r) \in \hat{\Sigma}_r$, $\delta'(q, (l_1, \ldots, l_r)) = q_0$ if for all $k \leq r$, $l_{j_k} = \diamond$ and $\delta'(q, (l_1, \ldots, l_r)) = q'$, if $\delta_P(q, (l_{j_1}, \ldots, l_{j_{ar}})) = q'$. Then we obtain $A_{P(x_{j_1}, \ldots, x_{j_{ar}})}$ by minimising the product of $A'_{P(x_{j_1}, \ldots, x_{j_{ar}})}$ and A_{D^r}. It is clear that the time required to build $A_{P(x_{j_1}, \ldots, x_{j_{ar}})}$ is also bounded by some polynomial of $\|AP\|^r$. Having a minimal automaton is needed in section 3.

The case of negation is closely related to automaton complementation which is simple for deterministic automata which we use. But a word in $\hat{\Sigma}_r^*$ is neither necessarily a convolution of words of $L(A_D)$, nor a convolution of words in Σ^*. Therefore the automaton for $\neg\psi$ is built from the complement of the automaton for ψ, followed by an on-the-fly product with A_{D^r}. Notice that we don't minimise this automaton; our results on complexity will still hold.

The case of conjunction is straightforward thanks to the fact that the free variables of the two formulas must be the same. The automaton is built as an on-the-fly product. We also do not need to minimise this inductively generated automaton.

The last case is $\varphi = \exists y.\psi(\bar{x}_r, y)$. By induction (as ψ is a subformula of φ) we build the $(r+1)$-variable automaton $A_\psi = (\hat{\Sigma}_{r+1}, Q_\psi, q_0, F_\psi, \delta_\psi)$. We assume that the track corresponding to variable y in A_ψ is the $(r+1)$-th (other cases are the same). We define from the $(r+1)$-variable automaton A_ψ by projection a *non-deterministic* r-variable automaton A'_φ that accepts representatives of solutions of φ. $A'_\varphi = (\hat{\Sigma}_r, Q_\varphi, Q_0, F_\varphi, \delta_\varphi)$ is built as follows: $Q_\varphi = Q_\psi$, the set of initial states Q_0 is the set of states reachable in A_ψ from q_0 by

transitions labelled in $\{\diamond\}^r \times \Sigma$, $F_\varphi = F_\psi$ and $\delta_\varphi(q,a) = \{q' \mid \exists b \in \Sigma \cup \{\diamond\}.\delta_\psi(q,(a,b)) = q'\}$.

We show the correctness of our construction. First we show that any word accepted by A'_φ is a convolution of words representing a solution of φ. Consider a word u that is accepted by A'_φ (see Fig. 1). Then there is an accepting run of u. Denote by q_1 the first state of the run (which is an initial state of A'_φ) and by q_2 the last state of the run ($q_2 \in F_\psi$). From this run we can get a word $w' \in \hat{\Sigma}^*_{r+1}$ (with $w'\downarrow_i = u\downarrow_i$ for any $i \leq r$) reaching q_2 from q_1 in A_ψ. By definition of the initial states of A'_φ, there is a word w'' in $(\{\diamond\}^r \times \Sigma)^*$ such that w'' reaches q_1 from q_0 in A_ψ. Thus $w''w'$ is accepted by A_ψ, which means it is a convolution of $r+1$ words in $L(A_D)$, so for any $i \leq r+1$, $(w''w')\Downarrow_i \in L(A_D)$. By definition of w' and w'', $(w''w')\Downarrow_i = u\Downarrow_i$ for any $i \leq r$ meaning u is a convolution of r words all in $L(A_D)$. We know that $([(w''w')\Downarrow_i])_{i\leq r+1}$ satisfies ψ, so $([w''w'\Downarrow_i])_{i\leq r} = ([u\Downarrow_i])_{i\leq r}$ satisfies $\exists y.\psi(\bar{x}_r, y)$. Thus u is a convolution of words in $L(A_D)$ that represent a solution of φ. We now show that A'_φ accepts any convolution of words in $L(A_D)$ that represent a solution of φ. Consider a solution of φ and take a representation u. There must exist a word w' such that the convolution of u and w' is accepted by A_ψ. Then u is also accepted by A'_φ. That concludes the proof of correctness of the construction of A'_φ an automaton accepting solutions of φ.

Finally we get A_φ by determinising A'_φ using the standard on-the-fly subset construction. Though in practice one can minimise this automaton, our complexity results still hold even if we don't.

3 Ehrenfeucht-Fraïssé relations for automata

Ehrenfeucht-Fraïssé equivalence relations are a general tool to establish upper and lower bounds on the complexity of the first-order theory over relational structures. They have been used for example extensively by Ferrante and Rackoff [5] to give some upper bounds for the decision procedure of several first-order logics. Let $\mathcal{A}$ be a relational structure with domain A. A set of Ehrenfeucht-Fraïssé equivalence relations for $\mathcal{A}$ is a ($\mathbb{N}^2$-indexed) family of relations $(E^r_m)_{m\in\mathbb{N},r\in\mathbb{N}}$ over A^r such that:

- $\bar{a}_r E^r_0 \bar{b}_r$ iff for any quantifier-free formula φ over r free variables: $\mathcal{A} \vDash \varphi(\bar{a}_r)$ iff $\mathcal{A} \vDash \varphi(\bar{b}_r)$
- Let $\bar{a}_r E^r_{m+1} \bar{b}_r$, then $\forall a_{r+1} \in A$, $\exists b_{r+1} \in A$ such that $(\bar{a}_r, a_{r+1}) E^{r+1}_m (\bar{b}_r, b_{r+1})$.

As a result any first-order formula with r free variables and quantifier-depth at most m cannot distinguish between tuples in the same E^r_m equivalence class. In [5] it is shown that in a first-order formula for several logics like Presburger Arithmetic quantifiers ranging over all elements of the domain can be restricted to finite subsets, hence obtaining space-constrained non-deterministic algorithms that exhaustively check the validity of these formulas with restricted quantification. The complexity of the decision procedures in [5] is closely related to that of deciding whether a predicate holds (usually simple) and the space required to enumerate these finite subsets, which depends on the size of the candidate b_{r+1}.

As we work on automatic presentations, the domain is a language and tuples of elements of the domain can also be seen as words. Thus we can consider the family of Ehrenfeucht-Fraïssé relations as a family of relations over languages and also impose that these relations are right-congruences allowing to relate equivalence classes with states of an automaton. This idea was used first by Ladner [12], working on monadic second-order and first-order logics on words, to deduce from the finiteness of the index the possibility to build a finite automaton. Recently Klaedtke [10] and then Eisinger [3] used this idea to give upper bounds on the size of automata for some (ω-)automatic structures. Our theorem below, not only bounds the size of automata but also allows us to establish an upper bound for the (time) complexity of the inductive construction of an r-variable automaton accepting solutions of a first-order

formula. This is possible since we can show that all automata inductively constructed satisfy the property that two words in the same equivalence class lead to the same state.

▶ **Theorem 5.** *Let $AP = (\Sigma, \mathcal{S}, A_D, A_=, (A_P)_{P\in\mathcal{S}})$ be a deterministic automatic presentation and (E^r_m) a family of* ***binary symmetric reflexive transitive relations*** *over $\hat{\Sigma}^*_r$ such that:*

1. *For any m, words of $\hat{\Sigma}^*_r$ that do not represent a convolution of words in Σ^* are alone in a same E^r_m equivalence class. The empty-word, ϵ, is alone in its E^r_m equivalence class.*
2. *Let uE^r_0v, if u is a convolution of r words in $L(A_D)$ then so is v and the r-tuples represented by u and v* ***satisfy the same atomic formulas*** *in the structure presented by AP.*
3. ***(back-and-forth)*** *If u is a convolution of r words in Σ^* and $uE^r_{m+1}v$, then for any $u_{r+1} \in \Sigma^*$, there exists $v_{r+1} \in \Sigma^*$ such that $\langle \overline{u\Downarrow}_r, u_{r+1}\rangle E^{r+1}_m \langle \overline{v\Downarrow}_r, v_{r+1}\rangle$.*
4. *The E^r_m are* ***right-congruence*** *relations: $uE^r_m v$ implies $\forall w \in \hat{\Sigma}^*_r$, $uwE^r_m vw$.*
5. *The* ***index*** *of E^r_m is* ***bounded*** *by $f(m+r)$, for some function f.*

Then the following holds: For any first-order formula φ over $\mathcal{S}$ with quantifier depth at most m and r free variables, the inductive construction of a deterministic r-variable automaton for φ builds an automaton with at most $f(m+r)$ states and can be done within time bounded by $c_1\|\varphi\| \left(\|AP\|^{m+r} f(m+r)\right)^{c_2}$ for some constants c_1 and c_2.

Proof. We first remark that hypothesis 2 can be generalised to: for any m, if $uE^r_m v$ and u is a convolution of r words in $L(A_D)$, then so is v and the r-tuples represented by u and v satisfy the same atomic formulas (and even the same r-variables formulas with quantifier depth at most m). We show that by applying m times hypothesis 3 (with a valid u_{r+1}), and then discarding the m new components.

The proof is by structural induction over formulas φ. Each formula has some quantifier depth m and some number r of free variables. The bound on the number of states is shown by proving inductively that any two words in $\hat{\Sigma}^*_r$ in the same E^r_m equivalence class reach the same state in the constructed automaton. Hence the number of states of these automata is bounded by $f(m+r)$. We choose the constants c_1 and c_2 as the maximum of the constants needed to bound the construction of the automata below. Sizes of automata are always bounded by a fixed polynomial of $\|AP\|^{m+r} f(m+r)$, since the alphabet is bounded by $\|AP\|^{m+r}$, and the number of states by $f(m+r)$.

We start with the **case of atomic formulas**, i.e. φ is of the form $P(x_{j_1}, \ldots, x_{j_r})$ where P is a predicate with arity ar and $(j_k)_{1\leq k\leq ar}$ a family of ar integers in $[1, r]$. $A_{P(x_{j_1},\ldots,x_{j_r})}$ is built as a minimal automaton. If u and v are E^r_m equivalent, according to hypothesis 4, for any $w \in \hat{\Sigma}^*_r$, $uwE^r_m vw$. Thus with generalised hypothesis 2, this means that uw represents a solution of $P(x_{j_1}, \ldots, x_{j_r})$ iff vw does, so $uw \in L(A_{P(x_{j_1},\ldots,x_{j_r})})$ iff $vw \in L(A_{P(x_{j_1},\ldots,x_{j_r})})$. The Myhill-Nerode theorem allows us to conclude that u and v reach the same state in $A_{P(x_{j_1},\ldots,x_{j_r})}$ as it is minimal. The time to build this automaton is bounded by a fixed polynomial of $\|AP\|^r$.

The case of negation is $\varphi = \neg\psi$. A_φ is built as a product automaton between the complement of A_ψ and A_{D^r}. Let $u, v \in \hat{\Sigma}^*_r$ with $uE^r_m v$. Hypothesis 4 implies that for any $w \in \hat{\Sigma}^*_r$, $uwE^r_m vw$; according to generalised hypothesis 2 this implies that uw is a convolution of words in $L(A_D)$ (and thus $uw \in L(A_{D^r})$) iff vw also is. As A_{D^r} is minimal by construction, the Myhill-Nerode theorem ensures that u and v reach the same state in A_{D^r}. As u and v reach the same state in A_ψ by induction hypothesis, they also reach the same state in the corresponding product automaton, which therefore has at most $f(m+r)$ states. By induction hypothesis, it takes time less than $c_1(\|\varphi\| - 1)(\|AP\|^{m+r} f(m+r))^{c_2}$ to build A_ψ. A_ψ has at most $f(m+r)$ states, and an alphabet of size smaller than $\|AP\|^r$. Thus any

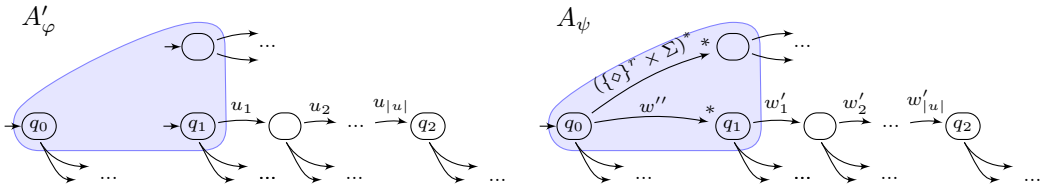


Figure 1 Getting a word $w''w' \in \hat{\Sigma}^*_{r+1}$ reaching q_2 in A_ψ from $u \in \hat{\Sigma}^*_r$ reaching q_2 from q_1 in A_φ.

manipulation over A_ψ will take time less than a fixed polynomial of $\|AP\|^{m+r} f(m+r)$, this includes its complementation and product with A_{D^r}.

The case of conjunction is $\varphi = \varphi_1 \wedge \varphi_2$. By induction hypothesis, $uE^r_m v$ implies that u and v reach the same state in A_{φ_1} and in A_{φ_2}, hence they will reach the same state in the product automaton (we don't even need to minimise A_φ to ensure this property). The time upper bound also holds as it takes total time less than $c_1(\|\varphi\|-1)(\|AP\|^{m+r} f(m+r))^{c_2}$ to build both A_{φ_1} and A_{φ_2} whose sizes are bounded by a fixed polynomial of $\|AP\|^{m+r} f(m+r)$, hence it takes a total time less than $c_1\|\varphi\|(\|AP\|^{m+r} f(m+r))^{c_2}$ to build A_φ.

The last case is $\varphi = \exists y.\psi(\bar{x}_r, y)$, where ψ has $r+1$ free variables and quantifier-depth at most $m-1$. Its automaton $A_\psi = (Q_\psi, q_0, F_\psi, \delta_\psi)$ is inductively built within time $c_1(\|\varphi\|-1)(\|AP\|^{m+r} f(m+r))^{c_2}$. We assume that the track corresponding to the variable y in A_ψ is the $(r+1)$-th. Let $A'_\varphi = (\hat{\Sigma}_r, Q_\varphi, Q_0, F_\varphi, \delta_\varphi)$ be the (non-deterministic) automaton as constructed in Section 2.3. We denote q_s the state of A_ψ reached by all words that are not convolutions of words in Σ^*. Let $uE^r_m v$. We show that u and v reach the same set of states in A'_φ. There are three cases: (1) u is not a convolution of words in Σ^*. Then nor is v and they obviously only reach the state q_s in A'_φ. (2) $u = \epsilon$, then $v = \epsilon$ and u and v are equal and clearly reach the same states. (3) $u \neq \epsilon$ (then $v \neq \epsilon$) and u is a convolution of words in Σ^* (then so is v). This first implies that both u and v can reach q_s in A'_φ as (for any $a \in \Sigma$) $\langle \overline{u\Downarrow}_r, a\diamond\rangle$ and $\langle \overline{u\Downarrow}_r, a\diamond\rangle$ are not convolutions of words in Σ^* so they both reach q_s in A_ψ. Assume now that u reaches a state $q_2 \neq q_s$ from q_1 (as depicted in Fig. 1). We can deduce w' and w'', such that $w'' \in (\{\diamond\}^r \times \Sigma)^*$ and for all $i \leq r$, $w'\downarrow_i = u\downarrow_i$, and $w''w'$ reaches q_2 from q_0 in A_ψ. As $w''w'$ does not reach q_s (in A_ψ), it is a convolution of words in Σ^* (induction hypothesis and hypothesis 1). Notice that $w''w'$ is the convolution of $(w\Downarrow_i)_{1\leq i\leq r}$ and $(w''w')\Downarrow_{r+1}$. According to hypothesis 3, there is a word $v' \in \Sigma^*$ such that $\langle \overline{v\Downarrow}_r, v'\rangle$ (the convolution of the $(v\Downarrow_i)_{1\leq i\leq r}$ and v') is E^{r+1}_{m-1} equivalent to $w''w'$. According to the induction hypothesis this implies $w''w'$ and $\langle \overline{v\Downarrow}_r, v'\rangle$ reach the same state in A_ψ, so there is a word reaching q_2 in A_ψ that is a convolution of the $(v\Downarrow_i)_{1\leq i\leq r}$ with another word in Σ^*, which means that v can also reach q_2 in A'_φ.

We have shown that any $u, v \in \hat{\Sigma}^*_r$ with $uE^r_m v$ reach the same set of states in A'_φ, hence by definition of the subset construction, they reach the same state in A_φ. Thus, A_ψ, A'_φ and A_φ each have at most $f(m+r)$ states over an alphabet bounded by $\|AP\|^{m+r}$ and it takes time polynomial w.r.t. the size of these automata to build A_φ from A_ψ, thus within time $c_1(\|AP\|^{m+r} f(m+r))^{c_2}$. That concludes the induction. ◀

Notice that we don't need to minimise any inductively-generated automaton during the construction. Furthermore remark that our approach only uses Ehrenfeucht-Fraïssé relations to prove an upper bound of the complexity of the automata construction (which might be much more efficient in particular cases), whereas Ferrante and Rackoff [5] need these relations to devise decision procedures.

4 Automata construction for structures of bounded degree

Automatic structures of bounded degree are structures whose uniform model-checking problem is known to be elementary [11]. Informally, a structure has bounded degree if there is a finite upper bound on the number of elements any element of the domain can be in relation with. We first formally define the necessary notions. The *Gaifman-graph* $G(\mathcal{A})$ of a relational structure $\mathcal{A} = (A, (P)_{P\in\mathcal{S}})$ is the graph $G(\mathcal{A}) = (A, \{(a,b) \in A \times A \mid \exists P \in \mathcal{S}\, \exists i,j.\, \exists \bar{a}_{ar_P} \in A^{ar_P}.\, a_i = a, a_j = b, \text{ and } P(\bar{a}_{ar_P}) \text{ holds}\})$. A structure has *bounded degree* if its Gaifman-graph has bounded degree, i.e. there exist a constant δ such that every node of the graph is adjacent to at most δ other nodes. The minimal such δ is called the *degree* of $\mathcal{A}$. An automatic presentation AP is of bounded degree, if $\mathcal{A}(AP)$ is of bounded degree. Then, the degree of AP is the same as the degree of $\mathcal{A}(AP)$. The following proposition is from [11].

▶ **Proposition 6.** *Let AP be an automatic presentation of bounded degree. Its degree is bounded by $2^{2^{\|AP\|^c}}$ for some constant c. If AP is injective, then its degree is bounded by $2^{\|AP\|^c}$ for some constant c.*

The following theorem is an application of Theorem 5.

▶ **Theorem 7.** *The construction of the automaton for injective deterministic automatic structures AP with bounded degree leads to an automaton whose size is bounded by $f(m+r) = 2^{2^{3^{m+r+c_3\cdot\|AP\|+2}}}$ within time $c_1\|\varphi\|(\|AP\|^{m+r} f(m+r))^{c_2}$ for some constants c_1, c_2 and c_3 independent of AP.*

To prove the theorem we have to give Ehrenfeucht-Fraïssé relations satisfying the hypotheses of Theorem 5. Let us fix for the rest of this section an injective deterministic automatic presentation $AP = (\Sigma, \mathcal{S}, A_D, A_=, (A_P)_{P\in\mathcal{S}})$ of bounded degree δ. Thanks to Proposition 6 we know that $\delta \leq 2^{\|AP\|^c}$. We can furthermore assume that the automata are minimal. Let Q_D be the set of states of A_D and Q_P the set of states of each A_P. We denote ar_P the arity of each predicate P, and $ar_M = \max_{P\in\mathcal{S}} ar_P$.

Using $\mathcal{A}(AP)$ we define a structure $\mathcal{A}(AP)_{sat}$, for which it will be easier to express Ehrenfeucht-Fraïssé relations satisfying all the hypotheses of Theorem 5. For example, to show right-congruence it will be necessary to be able to distinguish words leading to different states in the automata of AP.

$\mathcal{A}(AP)_{sat}$ is defined as the structure $(\Sigma^*, (P)_{P\in\mathcal{S}'})$ with the following predicates in the signature $\mathcal{S}'$: an "empty-word" monadic predicate denoted P_ϵ which holds exactly for the empty word, a monadic predicate $P_{D,q}$ for each state $q \in Q_D$ holding exactly for words that reach q in A_D and a predicate $P_{P,q}$ with arity r for each predicate P with arity r and each state $q \in Q_P$ that is not a sink state (i.e. with empty residual). $P_{P,q}$ holds exactly for r-tuples whose convolution reaches q in A_P.

▶ **Lemma 8.** *The degree of the structure $\mathcal{A}(AP)_{sat}$ is bounded by $\delta' = \delta \sum_{P\in\mathcal{S}} |Q_P|\, ar_P^2$.*

We prove this by contradiction: if x is in relation with too many words in $\mathcal{A}(AP)_{sat}$, too many ar_P-tuples containing x reach a state in A_P from which a final state can be reached. From this, we can deduce a word of $L(A_D)$ in relation with more than δ words in $\mathcal{A}(AP)$.

Before we define the Ehrenfeucht-Fraïssé relations we need the following definitions:

▶ **Definition 9.** For a relational structure $\mathcal{B}$ with domain B the Gaifman metric $d_\mathcal{B}(b_1, b_2)$ for $b_1, b_2 \in B$ is the distance between b_1 and b_2 in $G(\mathcal{B})$, that is the length of the shortest path connecting b_1 and b_2 in $G(\mathcal{B})$ (or $+\infty$ if b_1 and b_2 don't belong to the same connected

component). The $\mathcal{B}$-sphere of radius $d \in \mathbb{N}$ around $b \in B$ denoted by $\mathcal{S}_{\mathcal{B}}(b,d)$ is defined as the set $\{b' \in B \mid d_{\mathcal{B}}(b,b') \leq d\}$. We extend the notion of sphere around a point to spheres around r points, we call that the $\mathcal{B}$-neighbourhood of radius d around $\bar{b}_r$: for $\bar{b}_r \in B^r$ and $d \in \mathbb{N}$, $\mathcal{N}^r_{\mathcal{B}}(\bar{b}_r, d) = \bigcup_{1 \leq i \leq r} \mathcal{S}_{\mathcal{B}}(b_i, d)$. Finally, a $\mathcal{B}$-isomorphism ξ from $B_1 \subseteq B$ to $B_2 \subseteq B$ is a bijection that maps B_1 to B_2 such that for any predicate P of $\mathcal{B}$ with arity ar_P, and any ar_P-tuple $\bar{b}_{ar_P}$ of B_1, $P(b_1, \ldots, b_{ar_P})$ holds iff $P(\xi(b_1), \ldots, \xi(b_{ar_P}))$ holds. We will say that B_1 and B_2 are $\mathcal{B}$-isomorphic and write $B_1 \overset{\xi}{\simeq}_{\mathcal{B}} B_2$.

The Ehrenfeucht-Fraïssé relations that we define roughly state that r-tuples of words are equivalent when they have sufficiently large isomorphic neighbourhoods (i.e. of exponential radius) in the structure $\mathcal{A}(AP)_{sat}$.

▶ **Definition 10.** We define the equivalence relations E^r_m over $\hat{\Sigma}^*_r$ as follows: First we partition $\hat{\Sigma}^*_r$ in two disjoint subsets, V_r the set of words that are convolution of words in Σ^* and I_r the set of words that aren't. Then we define $u E^r_m v$ iff (1) u and v are in I_r, (2) or $u = v = \epsilon$ (3) or u and v are in $V_r \backslash \{\epsilon\}$ and $\mathcal{N}^r_{\mathcal{A}(AP)_{sat}}(\overline{u\Downarrow_r}, (3^m - 1)/2) \overset{\xi}{\simeq}_{\mathcal{A}(AP)_{sat}} \mathcal{N}^r_{\mathcal{A}(AP)_{sat}}(\overline{u\Downarrow_r}, (3^m - 1)/2)$ for some ξ such that $\xi(u\Downarrow_i) = v\Downarrow_i$.

It is clear that the relations E^r_m are symmetric, reflexive and transitive, and satisfy hypothesis 1 of Theorem 5. Due to space limitations, we just sketch here the proofs that this family of relations satisfy the other hypotheses of Theorem 5. To show it satisfies hypothesis 2 we essentially just need the fact that atomic formulas of $\mathcal{A}(AP)$ can be expressed as quantifier-free formulas in $\mathcal{A}(AP)_{sat}$. The back-and-forth property (hypothesis 3) is proved by exhibiting the v_{r+1} and extending the isomorphism. v_{r+1} is: (1) the image of u_{r+1} by the neighbourhood isomorphism, if u_{r+1} is "close" to $(\overline{u\Downarrow_r})$ (i.e. distance smaller than 3^m) (2) u_{r+1}, if it is "far" from the $(\overline{v\Downarrow_r})$ and the $(\overline{u\Downarrow_r})$, (3) some iterated preimage of u_{r+1} by the neighbourhood isomorphism, if u_{r+1} is in the neighbourhood of radius 3^m around $(\overline{v\Downarrow_r})$ but "far" from $(\overline{u\Downarrow_r})$. The closure of this relation by appending arbitrary suffix (hypothesis 5) crucially relies on the additional predicates provided by $\mathcal{A}(AP)_{sat}$.

Finally the following lemma states an upper bound on the index of the E^r_m relations:

▶ **Lemma 11.** *The index of E^r_m is bounded by $2^{2^{2^{g(m,r,ar_M,\delta,\|AP\|)}}}$ with $g(m,r,ar_M,\delta,\|AP\|) = (m+2).\log_2(3) + \log_2(\log_2(r)) + 2\log_2(ar_M) + \log_2(\log_2(\delta)) + \log_2(\log_2\|AP\|)$.*

This lemma can be proved noticing that as the degree of $G(\mathcal{A}(AP)_{sat})$ is bounded by δ' (Lemma 8), an $\mathcal{A}(AP)_{sat}$-neighbourhood of radius 3^m around r points has at most $r.\delta'^{3^{m+1}}$ elements. Thus there are at most $\prod_{P \in \mathcal{A}(AP)_{sat}} 2^{k^{ar_P}}$ non $\mathcal{A}(AP)_{sat}$-isomorphic k-elements sets. That concludes the picture of the proof of Theorem 7.

Notice that Theorem 7 only considers injective deterministic automatic presentations. Using Corollary 4.3 of [8] it is easy to see that an automatic presentation AP which is non-deterministic and not injective can be transformed into a deterministic injective presentation AP' such that $\|AP'\| \leq 2^{\|AP\|^c}$ for some constant c. Notice that the bound on the index of the E^r_m relations in Lemma 11 only depends exponentially on the size of the automatic presentation and it depends exponentially on its degree (which is bounded by a double exponential for a non-injective structure, see Proposition 6). Therefore we can obtain a deterministic automaton representing solutions of a formula φ in the structure $\mathcal{A}(AP')$ (which is isomorphic to $\mathcal{A}(AP)$) in triple exponential time. Therefore we obtain the following corollary improving the 3EXPSPACE upper bound of [11]. Moreover, we get easily in 3EXPTIME a non-deterministic automaton representing solutions in the structure $\mathcal{A}(AP)$.

▶ **Corollary 12.** *The model-checking problem for automatic presentations of bounded degree is in 3EXPTIME.*

5 Automata construction for Presburger Arithmetic

Presburger Arithmetic (PA) is the first-order theory over $\mathcal{A} = (\mathbb{Z}, +_{/3}, >_{/2})$, the structure over integers with addition and ordering. It was shown decidable [13] using quantifier elimination. Ferrante and Rackoff [5] gave the first definition of an Ehrenfeucht-Fraïssé relation over integers for PA. Büchi showed that PA was automatic [1] and Eisinger [3] showed that when using a suitable presentation based on most-significant digit first complement notation, Ferrante and Rackoff's relations are preserved by appending arbitrary suffixes allowing to obtain an upper bound on the size of the minimal automaton for a formula.

A common encoding of integer vectors is to use a binary representation and 0 (if the number is positive) or 1 (if it is negative) as padding symbols instead of $\diamond$. This leads to a non-injective presentation. Here, we use an injective automatic presentation of PA which is convenient for our purposes based on the binary most-significant digit first with complement notation: the alphabet is $\Sigma = \{0,1\}$ and valid encodings are in the language $D = \{0,1\} \cup 01\{0,1\}^* \cup 10\{0,1\}^*$. We denote by μ the isomorphism from elements of D to $\mathbb{Z}$. We have $\mu(0) = 0$, $\mu(1) = -1$ and $\mu(01w) = 2^{|w|} + \sum_{i=1}^{|w|} 2^{|w|-i} w[i]$, $\mu(10w) = -2^{|w|+1} + \sum_{i=1}^{|w|} 2^{|w|-i} w[i]$. It is easy to construct automata A_D with $L(A_D) = D$ and $A_>$ and A_+ for comparison and addition. Then, we get the injective deterministic automatic presentation for PA, $AP_{Pres} = (\Sigma, \{>, +\}, A_D, A_=, A_>, A_+)$ where $A_=$ accepts the identity relation over D.

The rest of the section is devoted to defining Ehrenfeucht-Fraïssé relations which satisfy the 5 hypotheses of Theorem 5. We first recall the Ehrenfeucht-Fraïssé relations of [3] for tuples of integers. We need to define inductively some families of integers and of sets of integers. Let B_m, B'_m, δ_m such that $B_0 = \{-2,-1,0,1,2\}$, $\delta_m = \operatorname{lcm} B_m$, $B'_m = \{\delta_m v/v' \mid v, v' \in B_m, v' \neq 0\}$ and $B_{m+1} = B_m \cup \{v + v' \mid v, v' \in B'_m\}$. Eisinger [3] defines an Ehrenfeucht-Fraïssé relation over tuples of integers inspired by Ferrante and Rackoff [5] as follows:

▶ **Definition 13.** ([3], Definition 1) For two k-tuples of integers $\bar{u}_k$ and $\bar{v}_k$ we define the equivalence relation F^r_m as $\bar{u}_k F^r_m \bar{v}_k$ iff for any i, $u_i \equiv v_i \mod \delta^2_m$ and for all $a_1, \ldots, a_r \in B_m$, and for all $c \in \mathbb{Z}$, with $|c| \leq (r+1)\delta^2_m$, $\sum_{i=1}^r a_i u_i + c \geq 0$ iff $\sum_{i=1}^r a_i v_i + c \geq 0$.

We adapt these relations (over integers) to relations over words of $\hat{\Sigma}^*_r$. This adaptation is slightly more involved than in [3] due to the presence of the padding symbol $\diamond$. Furthermore we have to distinguish convolutions of words according to which of their components can be ϵ or not. We partition $\hat{\Sigma}^*_r$ in three disjoint subsets: V_r the set of words that are convolution of words in $D \cup \{\epsilon\}$, S_r the set of words w that are convolution of words in Σ^* such that there is an i with $w\Downarrow_i \notin D \cup \{\epsilon\}$ and I_r the set of words that aren't convolutions of words of Σ^*. We further partition V_r and S_r into languages indexed by subsets of $[1,r]$: let $K \subseteq [1,r]$, we define $V_{r,K} = \{w \in V_r \mid w\Downarrow_i \neq \epsilon \text{ iff } i \in K\}$ and $S_{r,K} = \{w \in S_r \mid w\Downarrow_i \neq \epsilon \text{ iff } i \in K\}$. Clearly $V_r = \bigcup_{K \subseteq [1,r]} V_{r,K}$ and $S_r = \bigcup_{K \subseteq [1,r]} S_{r,K}$.

▶ **Definition 14.** We define a family of relations over words of $\hat{\Sigma}^*_r$. For $u, v \in \hat{\Sigma}^*_r$, $uE^r_m v$ iff:

- $u, v \in I_r$, or $u, v \in S_{r,K}$ for some $K \subseteq [1,r]$.
- $u, v \in V_{r,K}$ for some K (so if $i \in K$, $u\Downarrow_i$ and $v\Downarrow_i$ are in D and represent integers) and:
  - For all $i \in K$, $\mu(u\Downarrow_i) \equiv \mu(v\Downarrow_i) \mod \delta^2_m$
  - For all $b_1, \ldots, b_r \in B_m$, for all $c \in \mathbb{Z}$, $|c| \leq (r+1)\delta^2_m$, $c + \sum_{i \in K} b_i.\mu(u\Downarrow_i) \geq 0$ iff $c + \sum_{i \in K} b_i.\mu(v\Downarrow_i) \geq 0$

▶ **Lemma 15.** *E^r_m satisfies hypotheses 1 to 5 of Theorem 5, with $f(m+r) = 2^{2^{2^{c(m+r)}}}$, for some fixed c.*

Each hypothesis is proved similarly to the corresponding one of [3]. Thus, we have defined Ehrenfeucht-Fraïssé relations satisfying the 5 hypotheses of Theorem 5 and we obtain the following corollary.

▶ **Corollary 16.** *The inductive construction of an automaton A_φ representing all solutions of a Presburger Arithmetic formula φ is in 3EXPTIME.*

6 Conclusion and Perspectives

We have given a triple-exponential upper bound on the size of the automaton corresponding to the solutions of a first-order formula over automatic structures of bounded degree. An open problem is to find a matching lower bound. One can easily deduce a double-exponential lower bound from [11] and it might be possibly to adapt their proof of a 2EXPSPACE lower bound for the model-checking problem to obtain a formula and a structure for which the corresponding automaton must be of triple exponential size. Another interesting question is to study how our method can be extended to the case of tree automatic structures of bounded degree [11] as well as for ω-automatic structures.

Acknowledgement. We would like to thank Florian Horn for helpful comments.

References

1 J. Richard Büchi. Weak second-order Arithmetic and Finite Automata. *Zeitschrift fur Mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.

2 Antoine Durand-Gasselin and Peter Habermehl. On the Use of Non-deterministic Automata for Presburger Arithmetic. In *CONCUR*, volume 6269 of *LNCS*, pages 373–387. Springer, 2010.

3 Jochen Eisinger. Upper bounds on the automata size for integer and mixed real and integer linear arithmetic. In *CSL*, volume 5213 of *LNCS*, pages 431–445. Springer, 2008.

4 Calvin C. Elgot. Decision problems of finite automata design and related arithmetics. *Trans. Amer. Math. Soc.*, 98:21–51, 1961.

5 Jeanne Ferrante and Charles Weill Rackoff. *The computational complexity of logical theories*, volume 718 of *Lecture Notes in Mathematics*. Springer-Verlag, 1979.

6 Haim Gaifman. On local and non-local properties. In *Proc. of the Herbrand Symposium*, volume 107 of *Studies in Logic and the Foundations of Mathematics*, pages 105–135. Elsevier, 1982.

7 Bernard R. Hodgson. On direct products of automaton decidable theories. *Theoretical Computer Science*, 19(3):331–335, 1982.

8 Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In *Logical and Computational Complexity*, volume 960 of *LNCS*, pages 367–392. Springer, 1994.

9 Felix Klaedtke. Bounds on the Automata Size for Presburger Arithmetic. *ACM Trans. Comput. Logic*, 9(2):1–34, 2008.

10 Felix Klaedtke. Ehrenfeucht-Fraïssé goes automatic for real addition. *Inf. Comput.*, 208(11):1283–1295, 2010.

11 Dietrich Kuske and Markus Lohrey. Automatic structures of bounded degree revisited. *J. Symb. Log.*, 76(4):1352–1380, 2011.

12 Richard E. Ladner. Application of model theoretic games to discrete linear orders and finite automata. *Information and Control*, 33(4):281–303, 1977.

13 Mojżesz Presburger. Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen, in welchem die Addition als einzige Operation hervortritt. In *Comptes Rendus du Premier Congres des Mathématiciens des Pays Slaves*, pages 92–101, 1929.

Improved Bounds for Bipartite Matching on Surfaces

Samir Datta[1], Arjun Gopalan[2], Raghav Kulkarni[3], and Raghunath Tewari[4]

1 **Chennai Mathematical Institute, India**
sdatta@cmi.ac.in
2 **BITS Pilani, India**
arjun91@gmail.com
3 **LIAFA Paris 7 and LRI* Paris 11, France**
kulraghav@gmail.com
4 **Indian Institute of Technology – Kanpur, India**
rtewari@cse.iitk.ac.in

Abstract

We exhibit the following new upper bounds on the space complexity and the parallel complexity of the Bipartite Perfect Matching (BPM) problem for graphs of small genus:

(1) BPM in planar graphs is in UL (improves upon the SPL bound from Datta et. al. [7]);
(2) BPM in constant genus graphs is in NL (orthogonal to the SPL bound from Datta et. al. [8]);
(3) BPM in poly-logarithmic genus graphs is in NC; (extends the NC bound for $O(\log n)$ genus graphs from Mahajan and Varadarajan [22], and Kulkarni et. al. [19].

For Part (1) we combine the flow technique of Miller and Naor [23] with the double counting technique of Reinhardt and Allender [27]. For Part (2) and (3) we extend [23] to higher genus surfaces in the spirit of Chambers, Erickson and Nayyeri [4].

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases Perfect Matching, Graphs on Surfaces, Space Complexity, NC, UL

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.254

1 Introduction

1.1 Matching Problems in Graphs

A *matching* M in a graph G is a set of vertex disjoint edges. The end-points of the edges in M are said to be *matched*. A *perfect matching* in a graph G is a matching M such that every vertex of G is matched. See [21] for an excellent introduction to matching and related problems.

Historically, matching problems have played a central role in Algorithms and Complexity Theory. Edmond's *blossom* algorithm [10] for Max-Matching was one of the first examples of a non-trivial polynomial time algorithm. It had a considerable share in initiating the study of *efficient computation*, including the class P itself; Valiant's #P-hardness [30] for counting perfect matchings in bipartite graphs provided surprising insights into the counting complexity classes. The study of whether Perfect-Matching is parallelizable

* Supported by the French ANR Defis program under contract ANR-08-EMER-012 (QRAC project)

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 254–265
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany
SYMPOSIUM ON THEORETICAL ASPECTS OF COMPUTER SCIENCE

has yielded powerful tools, such as the *isolating lemma* [25], that have found numerous other applications.

The rich combinatorial structure of matching problems combined with their potential to serve as central problems in the field invites their study from several perspectives. The focus of this paper is on the space and parallel complexity of matching problems. The best known upper bound for Perfect-Matching (and other matching problems mentioned above) is *non-uniform* SPL [2] whereas the best hardness known is NL-hardness [5]. Unless otherwise specified, all circuit classes from now on are uniform (say L-uniform).

1.2 Matching Problems in Planar Graphs

A well known example where planarity is a boon is that of counting perfect matchings. The problem in planar graphs is in P [17] as opposed to being #P-hard in general graphs [30]. Counting perfect matchings in planar graphs can in fact be done in NC [31]; thus Perfect-Matching (Decision) in planar graphs is in NC. "Is the construction version of Perfect-Matching in planar graphs in NC?" remains an outstanding open question, whereas the bipartite planar case is known to be in NC [23], [22], [19], [7].

The space complexity of matching problems in planar graphs was first studied by Datta, Kulkarni, and Roy [7] where it was shown that Min-Wt-PM in bipartite planar graphs is in SPL. Kulkarni [18] shows that Min-Wt-PM in planar graphs (not necessarily bipartite) is NL-hard. The only known hardness for Perfect-Matching in planar graphs is L-hardness (cf. [6]). For bipartite planar graphs, nothing better than L-hardness is known.

Given a directed graph G and two vertices s and t in G, let Dir-Reach denote the problem of deciding if there exists a path from s to t in G. Dir-Reach is NL-complete. It turns out that Dir-Reach in planar graphs reduces (in log-space) to Perfect-Matching in bipartite planar graphs [6]; the former was proved to be in $\mathsf{UL} \cap \mathsf{coUL}$ by Bourke, Tewari, and Vinodchandran [3]. In this paper we show that Perfect-Matching in bipartite planar graphs is in UL, leaving the coUL bound as an intriguing open question.

1.3 Matching Problems in small genus graphs

Counting perfect matchings in graphs embedded on $O(\log n)$ genus surfaces is in NC (see Galluccio and Loebl [12]). Combining this with a rounding procedure from Goldberg, Plotkin, Shmoys, and Tardos [13], the authors of [22] and [19] obtain an NC algorithm for the decision and construction versions of BPM in $O(\log n)$ genus graphs. In [8], the result of [7] was extended to bipartite graphs of bounded genus and a tighter bound of $\mathsf{SPL} \subseteq \mathsf{NC}$ was obtained. We are able to improve these results using a technique of Miller and Naor [23] and its extension to higher genus graphs by Chambers, Erickson and Nayyeri [4].

1.4 Our Results

▶ **Theorem 1.1.** Perfect-Matching *in bipartite planar graphs is in* UL.

The result holds for both decision and construction versions of the problem. We build on two key algorithms: (1) Miller and Naor's algorithm [23] for perfect matching in bipartite planar graphs; (2) Reinhardt and Allender's [27] UL algorithm for shortest path in *min-unique* graphs: graphs with polynomially bounded edge-weights and having at most one minimum weight path between any pair of vertices. Miller and Naor reduce the Perfect-Matching (Decision) in planar graphs to the following problem in directed planar graphs: Neg-Cycle

(Decision) problem - given a directed graph with polynomially bounded edge-weights, decide whether or not the graph contains a negative weight cycle. The simple observation that this reduction works in log-space combined with the crucial observation that Neg-Cycle problem is in NL yields the somewhat surprising NL bound for perfect matching in bipartite planar graphs. While this upper bound matches the lower bound of NL for matching in bipartite graphs, we are able to improve it to UL by making efficient use of planarity. This brings it tantalizingly close to the best known upper bound of UL ∩ coUL for planar reachability. For the proof of the UL bound in part (a), we first provide a technical extension of (2) when the graph contains negative weight edges but no negative weight cycles. A simple but subtle combination of (1) and (2) then yields the desired result. As opposed to [19] and [7], our space bounded algorithms do not require determinant computation as a subroutine, instead we make use of a variant of planar reachability. However, for the weighted case we do not know how to improve upon the SPL bound in [7]. We also do not know how to improve on $\mathsf{L}^{\mathrm{C}_=\mathrm{L}}$ bound for maximum matching due to Hoang [15].

▶ **Theorem 1.2.** Perfect-Matching *in bipartite graphs of constant genus is in* NL.

▶ **Theorem 1.3.** Perfect-Matching *in bipartite graphs of* $(\log n)^{O(1)}$ *genus is in* NC.

Again the results hold for both decision and construction versions but we require that a *cellular* embedding of the graph be given as part of the input. We adapt the approach of Chambers et. al. [4] in the context of the flow instance corresponding to the perfect matching problem. Chambers, Erickson and Nayyeri [4] extend the techniques of Miller and Naor to reduce the search space of a max-s,t-flow on a surface. In particular, for genus g surface they can formulate the flow problem as a linear program in only $O(g)$ variables. We show that a flow instance corresponding to perfect matching in a bipartite graph embedded on a surface is a yes instance exactly when we can send flows along the $2g$ *basis cycles* such that the residual graph has no negative cycle. Moreover, if we start from a Perfect-Matching instance then the flows must be integral and polynomially bounded. Thus exhaustive search in the $2g$-dimensional space yields an NL algorithm for the existence of a Perfect-Matching when g is a constant. We believe that this NL bound can be improved to UL.

For poly-logarithmic genus, the ellipsoid method yields an NC bound. Our observation is that the separation oracle, the problem of determining if a weighted directed graph contains a negative cycle, can be implemented in NC. For the construction version, we use the rounding procedure of Goldberg, Plotkin, Shmoys and Tardos [13] to obtain an integral solution in NC from the fractional solution coming from the ellipsoid algorithm.

We also consider Even-Path problem: deciding whether or not there is a directed simple path of even length between two specified vertices. Even-Path is NP-complete [20] but restricted to planar graphs it is in P [26]. For directed acyclic graphs (DAGs), the problem is NL-complete. The Even-Path problem can be viewed as a relaxation of the Red-Blue-Path problem - given a directed graph with edges colored Red or Blue, decide whether or not there is a (simple) path between two specified vertices such that consecutive edges in the path are of different colors. The Red-Blue-Path problem is known to be NL-complete for planar DAGs [18]. This provides context for the following theorem.

▶ **Theorem 1.4.** Even-Path *in planar DAG is in* UL.

The hope is that the proof of this theorem contains the seeds for a proof showing that Red-Blue-Path for planar DAGs is in UL which would imply that NL collapses to UL. It is worth noting that our proof of the UL bound for Even-Path in planar DAGs combines two different *deterministic isolation* techniques ([3], [15]).

1.5 Organization

Section 2 contains preliminaries. Section 3 contains the UL bound for bipartite planar graphs. Section 4 contains the results for higher genus graphs. Section 5 contains UL bound for Even-Path problem in planar DAG. Section 6 contains some open ends.

2 Preliminaries

2.1 Matching Problems

We consider the following computational problems related to matching:

- Perfect-Matching (Decision) : decide if G contains a perfect matching.
- Perfect-Matching (Construction) : construct a perfect matching in G (if exists).
- Min-Wt-PM (Decision) : given G together with edge-weights $w : E(G) \to \mathbb{Z}$ such that $|w(e)| \leq n^{O(1)}$, and an integer k - decide if G contains a perfect matching of weight at most k.
- Max-Matching (Decision) : given G and an integer k, decide if G has a matching of cardinality at least k.
- UPM (Decision) : decide if G has a unique perfect matching

2.2 Space Complexity Classes

See the monograph by Vollmer [32] for definitions of standard circuit complexity classes. It is known that $\mathsf{UL} \subseteq \mathsf{NL} \subseteq \mathsf{NC} \subseteq \mathsf{P}$ and $\mathsf{UL} \subseteq \mathsf{SPL}$. It is also known that $\mathsf{SPL} \subseteq \oplus\mathsf{L} \subseteq \mathsf{NC}$. NL and SPL as well as NL and $\oplus$L are not known to be comparable.

2.3 Flow Terminology

Here we rephrase the terminology used in [23]. An undirected *edge* is a two element unordered set $\{u, v\}$ such that $u, v \in V$. An undirected graph $G = (V, E)$ consists of a set V of *vertices* and a set E of *undirected edges*. An *arc* is an ordered tuple $(u, v) \in V \times V$. A directed graph $\overrightarrow{G} = (V, \overrightarrow{E})$ consists of a set V of *vertices* and a set $\overrightarrow{E} \subseteq V \times V$ of arcs. Given an undirected graph $G = (U, V)$, its directed version is a directed graph $\overleftrightarrow{G} = (V, \overleftrightarrow{E})$ where $\overleftrightarrow{E} := \{(u, v) \mid \{u, v\} \in E\}$.

A *capacity-demand graph* is a triple (G, c, d) where $G = (V, E)$; every arc $(u, v) \in \overleftrightarrow{E}$ is assigned a real value $c(u, v)$ called the *capacity* of the arc and every vertex $v \in V$ is assigned a real value $d(v)$ called the *demand* at the vertex. A *pseudo-flow* in a capacity-demand graph (G, c, d) is a function $f : \overleftrightarrow{E} \to \mathbb{R}$ such that: (i) for every arc $(u, v) \in \overleftrightarrow{E}$, we have: (skew-symmetry) $f(u, v) = -f(v, u)$, and (ii) for every vertex $v \in V$, we have: (demands met) $\sum_{w \in V:(v,w) \in \overleftrightarrow{E}} f(v, w) = d(v)$. A *flow* in a capacity-demand graph (G, c, d) is a function $f : \overleftrightarrow{E} \to \mathbb{R}$ such that: (a) f is a pseudo-flow in (G, c, d); (b) for every $(u, v) \in \overleftrightarrow{E}$, we have: (capacity constraints satisfied) $f(u, v) \leq c(u, v)$. A *zero-demand graph* (G, c) is a capacity-demand graph in which the demand at every vertex is zero.

For a description of other graph theoretic terminology (such as walk, dual graph etc.), we refer the reader to Diestel's excellent text [9].

2.4 Main Lemmas from Miller and Naor [23]

▶ **Definition 2.1** (Directed Dual)**.** Let G be a planar graph. Fix an embedding of G in the plane. Let G^* denote the dual of G with respect to the fixed embedding. The directed

dual of G is the directed version of G^* denoted by $\overleftrightarrow{(G^*)}$. The arcs of $\overleftrightarrow{G}$ and that of $\overleftrightarrow{(G^*)}$ are in one to one correspondence. If $e = \{u, v\}$ is an edge in G with a directed version (u, v) and $e^* = \{x^*, y^*\}$ is the corresponding dual edge in G^* then in $\overleftrightarrow{(G^*)}$ the directed edge that corresponds to (u, v) is directed (x^*, y^*) where x^* is the face that lies to the *left* of the directed edge (u, v).

▶ Proposition 2.2 (folklore, see for instance [23]). Let (G, c) be a zero-demand graph. Let f be a flow in (G, c). If $C^* = (e_1^*, \ldots, e_k^*)$ is a directed cycle in $\overleftrightarrow{(G^*)}$, then

$$\sum_{e \,:\, e^* \in C^*} f(e) = 0.$$

▶ **Lemma 2.3** (Miller, Naor [23]). *Let (G, c) be a zero-demand planar graph, then: there exists a flow in (G, c) $\iff$ $\overleftrightarrow{(G^*)}$ has no negative weight cycle with respect to weights c.*

3 Bipartite Planar Matching: The UL Bounds

Suppose we have a directed graph G with polynomially bounded weights on its edges. The weights could be positive or negative. Let s be a fixed vertex in G. Let $d(u, v)$ denote the length of the minimum length path from u to v. whenever defined. Notice that these definitions are conditional on the non-existence of negative cycles and we show how to deal with these cases below.

$$V_k := \{v \mid d(s, v) \leq k\}.$$

Let $\text{dist}_k^w(u, v)$ denote the weight of the minimum weight walk (with respect to weights w) of length at most k from u to v. Note that $\text{dist}_k^w(u, v)$ could be negative. We define,

$$\Sigma_k^w := \sum_{v \in V_k} \text{dist}_k^w(s, v).$$

We use an extension of [27] to compute $\text{dist}_k^w(s, v)$ and V_k in UL. We pause to note that the technique of [27], called double counting in [27], is a generalization of the inductive counting technique used by Immerman [16] and Szelepcsényi [28] to show that NL = coNL. We combine this UL algorithm with Miller and Naor's algorithm via Weighting Scheme A in Section 5 to obtain the UL bound for perfect matching in bipartite planar graphs.

We need an extension of [27], when the graph contains negative weight edges but no negative weight cycles. We call this extension (Algorithm 2) as the *Extended-RA Algorithm* . Following lemmas are simple consequences of the Extended-RA Algorithm and *min-uniqueness* achieved via generalized BTV weights (Weighting Scheme A).

The weighting scheme A

Weighting scheme A is a generalization of the weight function in [3] to planar graphs. In other words, given a directed planar graph G, we construct a log-space computable edge weight function with respect to which any simple cycle in G has non-zero weight. Tewari and Vinodchandran [29] give a log-space construction of such a weight function by an application of Green's Theorem. We give an alternate procedure (see Algorithm 1) that achieves the same result.

Input : A planar graph G
Output : An edge weight function w_A such that for any simple cycle C in G $w_A(C) \neq 0$

1 Compute a spanning tree T in G;
2 For any arc $e \in \overleftrightarrow{T}$, set $w_A(e) = 0$;
3 Let R denote the spanning tree in G^* consisting of the edges that do not belong to T. Fix a root r for R (say the unbounded face) and let $\overrightarrow{R}$ denote the orientation of R where each edge is oriented towards the root;
4 An arc $e^* = (u, v) \in \overrightarrow{R}$ separates the tree R into two subtrees. Let α_u denote the number of vertices in the subtree containing u. Set $w_A(u, v) = \alpha_u$ and $w_A(v, u) = -\alpha_u$;
5 Set $w_A(e) = w_A(e^*)$ for every $e \in E(G)$ where e^* is the (directed) dual edge of e;

Algorithm 1: Weighting Scheme A

▶ **Lemma 3.1** (adaptation of [3]). *With respect to the weight function w_A the absolute value of the sum of the weights of the arcs along any simple directed cycle is equal to the number of faces in the interior of the cycle.*

Proof of Lemma 3.1: For a simple cycle C of G, let us define the weight of C, $w(C)$, to be the sum of weights of the edges lying along C in clockwise order. Note that w_A is skew symmetric. Thus clockwise and anti-clockwise weights of the cycle C are the same in absolute value but opposite in sign. We are denoting the clockwise weight of C by $w(C)$.

It suffices to show that for a facial cycle F of G, $w(F) = +1$. This is because for a simple cycle C:

$$w(C) = \sum_{F \in \text{Interior}(C)} w(F).$$

But $w(F)$ equals the sum of the weights of dual edges (in G^*) outgoing from the dual vertex $F^* \in V(G^*)$, so it suffices to show that for every vertex $u \in V(G^*)$:

$$\sum_{v:(u,v)\in E(G^*)} \alpha_v = +1.$$

Observe that the number of nodes in the subtree rooted at u is one more than sum of the number of vertices in the subtrees rooted at v for various v, such that (u, v) is a dual edge. This, together with the skew symmetry of the weights $w_A(u, v)$, completes the proof. □

Input : A directed graph G on n vertices; edge-weights $w : E(G) \to \mathbb{Z}$ such that $|w(e)| \leq n^{O(1)}$; $s, v \in V(G)$; and an integer t
Output : $\text{dist}_t^w(s, v)$

1 Initialize $V_0 \leftarrow \{s\}$ and $\Sigma_0^w \leftarrow 0$;
2 **for** $k = 1$ *to* t **do**
3 | Compute $(|V_k|, \Sigma_k^w)$ from $(|V_{k-1}|, \Sigma_{k-1}^w)$;
4 **end**
5 Compute $\text{dist}_t^w(s, v)$ from $(|V_t|, \Sigma_t^w)$ and output;

Algorithm 2: Extended-RA Algorithm (adapted from [27])

▶ **Lemma 3.2.** *Given a directed planar graph with polynomially bounded weights w on its arcs such that there are no negative weight cycles, the shortest distance* $\text{dist}^w(u,v)$ *between any pair of vertices with respect to weights w can be computed in* UL.

▶ **Lemma 3.3.** *Given a directed planar graph with polynomially bounded weights w on its arcs, deciding whether or not the graph contains a negative weight cycle is in* coUL.

Input : A bipartite planar graph G
Output : A perfect matching in G if one exists; else reject

1 Construct a capacity-demand graph (G, c, d) as follows: for each vertex $v \in A$, set $d(v) = 1$ and for each vertex $v \in B$, set $d(v) = -1$. For $u \in A, v \in B$, set $c(u,v) = 1$ and $c(v,u) = 0$;
2 Construct a pseudo-flow f' in (G, c, d) (see [23]);
3 Construct a zero-demand graph $(G, c - f')$;
4 Run Extended-RA Algorithm on $\overleftrightarrow{(G^*)}$ with weights $w = n^4(c - f') + btv$ to compute the shortest distances $\text{dist}_n^w(u,v)$ in $\overleftrightarrow{(G^*)}$, where btv denotes generalized BTV weights (Weighting Scheme A defined in the beginning of this section);
5 Compute $\text{dist}_n^{c-f'}(u,v)$ in $\overleftrightarrow{(G^*)}$ from the above by ignoring the lower order weights from btv;
6 Run Miller and Naor's algorithm to compute f (see algorithm in Section 5.1 in [23]);
7 If f is a flow then for $u \in A$ and $v \in B$, output "u is matched to v" $\iff f(u,v) = 1$;
8 otherwise reject and output "No perfect matching";

Algorithm 3: UL algorithm for Perfect-Matching in bipartite planar graphs

Combining Algorithm 2 with Miller and Naor's algorithm, we obtain the UL algorithm (Algorithm 3) for Perfect-Matching in bipartite planar graphs.

▶ **Theorem 3.4.** *(Theorem 1.1) In bipartite planar graphs, both the decision as well as the construction versions of the* Perfect-Matching *are in* UL.

Proof. The correctness of the above algorithm follows from [23]. To see the UL bound, note that the Extended-RA algorithm computes dist_n^w correctly along a unique path assuming min-uniqueness of the weights. If there are no negative weight cycles then the generalized BTV weights (Section 5) guarantee min-uniqueness.

Thus, if there are no negative weight cycles in $\overleftrightarrow{(G^*)}$ then we obtain a valid flow and a perfect matching along the unique accepting path. Otherwise, we realize that f is not a valid flow and reject. ◀

We also obtain the following corollary on similar lines

▶ **Corollary 3.5.** *Single-source, single-sink maximum flow problem in planar networks with polynomially bounded capacities is in* L^{UL}.

4 Bipartite Perfect Matching in higher genus graphs

We need G to be given together with its *cellular* embedding [4] on a surface of genus g. Every graph admits a cellular embedding as the embedding on the minimal genus surface

is always cellular [24]. We also need the $2g$ *basis* cycles in G explicitly given to us. The advantage of cellular embedding of G is that every vertex of G corresponds to a face in the dual graph G^* and vice versa. Let G be a graph with a cellular embedding on a surface of genus g and let $C_1, C_2, \ldots, C_{2g}$ be the basis cycles. Using Steps 1, 2, and 3 of Algorithm 3, we first obtain a multiple source multiple sink flow problem and then transform it to a zero demand instance. None of these reductions use planarity. Let (G, c) denote the zero-demand instance associated with G with capacity function c. We fix an arbitrary orientation for each C_i. For $i = 1, \ldots, 2g$, let F_i denote the flow that is zero everywhere outside C_i, i.e., $F_i(e) = 0$ if $e \notin C_i$ and for each $e \in C_i$ the flow value is f_i, i.e., $F_i(e) = f_i$ Let $f = (f_1, \ldots, f_{2g})$ and let $c - f$ denote the graph with the weight of edge e defined as $c(e) - \sum_i F_i(e)$.

The following is a generalization of Lemma 2.3 in Section 2 (same as Lemma 4.1 in [23]) for higher genus graphs with cellular embeddings. After we obtained the proof of this lemma, we learned that a similar lemma is already noted by Chambers et al. [4].

▶ **Lemma 4.1.** *The zero demand instance (G, c) admits a valid flow if and only if there exists $f_1, \ldots, f_{2g}$ such that the dual graph G^* with weights $c - f$ has no negative cycles.*

Moreover: if the capacities c are integral then we can assume f_i to be integral.

Proof. Analogous to the proof of Lemma 3.1 in [4].

If (G, c) admits a valid flow F then we fix $2g$ basis cycles $C_1^*, \ldots, C_{2g}^*$ in the dual with an arbitrarily chosen orientation and we take $f_i = \sum_{e \in C_i^*} F(e)$. We need that C_i^* crosses C_i exactly once, and C_i^* does not cross C_j if $j \neq i$. We *claim* that this choice leaves no negative cycles in the dual with respect to weights $c - f$ (cf. Lemma 3.1 in [4]).

If there exists f such that there are no negative cycles in the dual with respect to weights $c - f$ then using the shortest distance in dual (proof of Lemma 4.1 in [23]) we can get a valid flow in the zero-demand instance. Here we use the fact that the embedding is cellular and hence every vertex corresponds to a cycle in the dual; and the flow obtained by the shortest distance in the dual sums up to zero on every cycle in the dual. ◀

We use the fact that if (G, c) admits a valid flow then there exists f such that the values of f_i are at most $c_{max} \cdot n$, where c_{max} is the maximum absolute value of the capacites to obtain the following.

▶ **Theorem 4.2.** *Given a bipartite graph G together with a cellular embedding on a constant genus surface,* Perfect-Matching *(Decision + Construction) in G is in* NL.

▶ **Theorem 4.3.** *Given a bipartite graph G together with a cellular embedding on a surface of poly-logarithmic genus,* Perfect-Matching *(Decision + Construction) in G is in* NC.

Proof. Note that Lemma 4.1 reduces the decision version of the Bipartite Perfect Matching problem to the problem of solving the feasibility of a linear program in variables $f_1, \ldots, f_{2g}$ with the linear constraints that every cycle in the dual is non-negative with respect to weights $c - f$. We use ellipsoid method to solve this problem.

The crucial observation is that the separation oracle for this problem is in NC. The separation oracle in our context is, given a weighted graph, the problem of determining whether or not it contains a negative cycle. This problem is equivalent to checking if all pair shortest paths are well-defined (because otherwise vertices lying on a negative cycle will have negative shortest paths to themselves). Thus a parallelized version of Floyd-Warshall which runs in NC even when the weights are exponential [14] is sufficient for our purpose.

The running time of the algorithm modulo the separation oracle is polynomial in the number of variables and hence in $g^{O(1)}$ time. This yields an NC algorithm for the decision version for poly-log genus graphs, given their embedding in the required form.

The construction version is also in NC,: A solution to the linear program in the $2g$ variables naturally translates to a point inside the Perfect Matching Polytope of G [19]. Pulling back a point from $\mathbb{R}^{2g}$ to $\mathbb{R}^{|E(G)|}$ can be accomplished in L via an argument similar to the proof of Lemma 4.1. An NC procedure to obtain a Perfect Matching, given a point inside the Perfect Matching Polytope is described in [13] (also see Section 3 in [19]). ◀

5 Even-Path in planar DAG is in UL

▶ **Definition 5.1** (Red-Blue-Path)**.** Given a directed graph with each edge colored either Red or Blue, a *Red-Blue-Path* from s to t is a (simple) directed path from s to t such that consecutive edges are of different colors. The RED-BLUE-PATH problem is to decide if there is a Red-Blue-Path from s to t.

▶ **Definition 5.2** (Even-Path)**.** Given a directed graph and two nodes s and t, an *Even-Path* from s to t is a (simple) directed path from s to t containing even number of edges. The EVEN-PATH problem is to decide if there is an Even-Path from s to t.

▶ **Theorem 5.3** ([18])**.** RED-BLUE-PATH *in planar DAGs is* NL*-complete.*

In this section, we prove that the EVEN-PATH problem (which can be viewed as a relaxation of the RED-BLUE-PATH problem as a path starting with say Red edge and ending with say Blue edge is always of even length) in planar DAG is in fact in UL. Our proof involves a combination of two different isolation techniques that are currently available.

▶ **Lemma 5.4.** *Let G be a planar DAG and u and v be any two vertices in G. Then with respect to the weight function w_A, (a) if P_1 and P_2 are two minimum weight Even-Paths from u to v, then $P_1 \oplus P_2$ (the symmetric difference between the sets of edges of P_1 and P_2) divides the plane into at most two bounded regions; (b) no three minimum weight Even-Paths from u to v share a common vertex w other than u and v, such that the path segments between the vertices u and w and between w and v are not identical. (c) there are at most $2n^4$ minimum weight Even-Paths from u to v.*

Proof. (a) For the sake of contradiction let C_1, C_2 and C_3 be any three bounded regions of $P_1 \oplus P_2$. Let P_{ij} be the restriction of the i-th path to the j-th region for $i \in \{1,2\}$ and $j \in \{1,2,3\}$. Observe that $w_A(P_{1j}) \neq w_A(P_{2j})$ since C_j is a simple cycle and by Lemma 3.1 we have that $w_A(C_j) \neq 0$. Now the parity of the lengths of the path segments $P_{1,j}$ and $P_{2,j}$ are different since if they were the same, we could replace the higher weighted segment with the lower weighted one and get an even length path of lesser weight. This implies that $|C_1| + |C_2| + |C_3|$ is odd since each $|C_i|$ is odd. Let $P'_i = \bigcup_j P_{ij}$ for $i \in \{1,2\}$. Therefore either $|P'_1|$ is odd or $|P'_2|$, but not both. Without loss of generality lets assume $|P'_1|$ is odd. For each j pick the path segment between P_{1j} and P_{2j} that has lesser weight to create a set say P'. Now $w_A(P')$ is strictly smaller than both $w_A(P'_1)$ and $w_A(P'_2)$. If $|P'|$ is odd then replace P'_1 with P' and if $|P'|$ is even then replace P'_2 with P' to get a path of smaller weight and same parity. This is a contradiction. Thus $P_1 \oplus P_2$ has at most two bounded regions.

(b) Let P_1, P_2 and P_3 be three minimum weight paths from u to v that share a common vertex (say w) such that the segments of each of the three paths between the vertices u and w and between w and v are distinct. In other words, if P'_i and P''_i are the segments of P_i between the vertices u and w and between w and v respectively (for $i \in \{1,2,3\}$), then $\{P'_i\}$ are pairwise non-identical and so are $\{P''_i\}$. There exists at least two path segments between P'_1, P'_2 and P'_3 whose lengths have the same parity. Without loss of generality assume its P'_1 and P'_2. Now if $w_A(P'_1) \neq w_A(P'_2)$ then since they have the same parity we can pick the

lesser weight path between P_1' and P_2' and similarly the lesser weight path between P_1'' and P_2'' and append them to get an even path of weight less than either that of P_1 or P_2 from u to v. Thus we can assume $w_A(P_1') = w_A(P_2')$. By Lemma 3.1, this implies that $P_1' \oplus P_2'$ as at least two bounded regions. Moreover since P_1'' and P_2'' are also not identical, therefore $P_1'' \oplus P_2''$ has at least one one bounded region. Thus $P_1 \oplus P_2$ has at least 3 bounded regions, thus contradicting part (a).

(c) Let a, b, c and d be four vertices in G and let $\mathcal{P}_{a,b,c,d}$ be the set of all minimum weight even length paths from u to v that pass through the vertices a, b, c and d in that order and are vertex disjoint between the vertices a and b and between the vertices c and d respectively. Then by part (b), $\mathcal{P}_{a,b,c,d}$ will have at most 2 paths. Since the total number of such tuples is at most n^4, therefore the number of minimum weight, even length u-v paths is bounded by $2n^4$. ◀

Constructing an auxiliary graph

Construct a directed (multi)graph G' from G as follows: the vertex set of G' is the vertex set of G. An edge (v_i, v_j) is in G' if and only if there exists a vertex v_k in G and the edges (v_i, v_k) and (v_k, v_j) are in G. The weight w of an edges in G' is the sum of the weights of the corresponding two edges in G.

Now Lemma 5.5 follows by definition of G' and part (c) of Lemma 5.4.

▶ **Lemma 5.5.** *(a) G has a directed Even-Path from u to v if and only if G' has a directed path from u to v; (b) the number of minimum weights paths from u to v in G' with respect to w_A is at most $2n^4$.*

Weighting scheme B

Our weighting scheme B is based on a well known hashing scheme based on primes, due to Fredman, Komlós and Szemerédi [11].

▶ **Lemma 5.6** ([11]). *Let c be a constant and S be a set of n-bit integers with $|S| \leq n^c$. Then there is a c' and a $c' \log n$-bit prime number p so that for any $x \neq y \in S$ $x \not\equiv y \pmod{p}$.*

Hoang used this scheme to give better upper bounds for Perfect-Matching in certain classes of graphs [15]. Aduri, Tewari and Vinodchandran showed that reachability in graphs where the number of paths from s to any vertex is bounded by a polynomial is in UL, by applying this hashing scheme. We use Lemma 5.6 here to define a weight function with respect to which G' is min-unique.

Let p_i be the i^{th} prime number. Consider the lexicographical ordering of the edges of G' and denote the j^{th} edge in this ordering by e_j. Define the i^{th} weight function (for $1 \leq i \leq q(n)$ and an appropriate polynomial $q(n)$ dictated by Lemma 5.6), $w_{B_i}(e_j) = 2^j \pmod{p_i}$.

▶ **Lemma 5.7** (Adapted from [1]). *There exists an $i \leq q(n)$ such that the graph G' with respect to the weight function $W_i = w_A \cdot n^{10} + w_{B_i}$ is min-unique.*

Proof. Let $\mathcal{P}_v$ be the set of minimum weight paths from s to a vertex v in G', with respect to w_A. Then by Lemma 5.5, $|\mathcal{P}_v|$ is bounded by $2n^4$. It follows from Lemma 5.6 that with respect to some w_{B_i}, all paths in $\bigcup_v \mathcal{P}_v$ will have distinct weights. Therefore G' is min-unique with respect to W_i for some i. ◀

For each $i \in [q(n)]$, check if G' is min-unique with respect to W_i or not. Once we have an appropriate i, we can decide reachability in G' in UL [27]. By Lemma 5.5 a path in G' corresponds to an EvenPath in G and thus we have Theorem 5.8.

▶ **Theorem 5.8.** *(Theorem 1.4)* Even-Path *in planar DAGs is in* UL.

6 Open Ends

Is Neg-Cycle (Decision) in planar graphs in UL? Is Odd-Cycle in planar graphs in ⊕L? Is Perfect-Matching (Decision) in bipartite planar graphs in coUL? Is Min-Wt-PM in bipartite planar graphs in NL? Is Max-Matching in bipartite planar graphs in NL?

Acknowledgement

We would like to thank Prajakta Nimbhorkar for discussion in the initial stages of the work, in particular for pointing out that the Neg-Cycle problem is in NL. We would like to thank V. Vinochandran for pointing out references [11] and [1] which are crucially used in the proof of Theorem 5.8. The second author has been supported by Microsoft Research India Travel Grant to attend the conference.

References

1 Pavan Aduri, Raghunath Tewari, and N. V. Vinodchandran. On the power of unambiguity in logspace. Technical Report TR10-009, Electronic Colloquium on Computational Complexity, 2010.

2 Eric Allender, Klaus Reinhardt, and Shiyu Zhou. Isolation, matching, and counting: Uniform and nonuniform upper bounds. *Journal of Computer and System Sciences*, 59:164–181, 1999.

3 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.

4 Erin W. Chambers, Jeff Erickson, and Amir Nayyeri. Homology flows, cohomology cuts. In *STOC*, pages 273–282, 2009.

5 Ashok K. Chandra, Larry Stockmeyer, and Uzi Vishkin. Constant depth reducibility. *SIAM Journal on Computing*, 13(2):423–439, 1984.

6 Samir Datta, Raghav Kulkarni, Nutan Limaye, and Meena Mahajan. Planarity, determinants, permanents, and (unique) matchings. *ACM Trans. Comput. Theory*, 1(3):1–20, 2010.

7 Samir Datta, Raghav Kulkarni, and Sambuddha Roy. Deterministically isolating a perfect matching in bipartite planar graphs. *Theory of Computing Systems*, 47:737–757, 2010. 10.1007/s00224-009-9204-8.

8 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.

9 Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, August 2005.

10 J. Edmonds. Paths, trees and flowers. *Canad. J. Math.*, 17:449–467, 1965.

11 Michael L. Fredman, János Komlós, and Endre Szemerédi. Storing a sparse table with 0(1) worst case access time. *J. ACM*, 31:538–544, June 1984.

12 Anna Galluccio and Martin Loebl. On the theory of pfaffian orientations. i. perfect matchings and permanents. *Electr. J. Comb.*, 6, 1999.

13 Andrew V. Goldberg, Serge A. Plotkin, David B. Shmoys, and Éva Tardos. Using interior-point methods for fast parallel algorithms for bipartite matching and related problems. *SIAM J. Comput.*, 21(1):140–150, 1992.

14 Yijie Han, Victor Y. Pan, and John H. Reif. Efficient parallel algorithms for computing all pair shortest paths in directed graphs. *Algorithmica*, 17(4):399–415, 1997.

15 Thanh Minh Hoang. On the matching problem for special graph classes. In *IEEE Conference on Computational Complexity*, pages 139–150, 2010.

16 Neil Immerman. Nondeterministic space is closed under complementation. *SIAM J. Comput.*, 17(5):935–938, 1988.

17 P. W. Kasteleyn. Graph theory and crystal physics. *Graph Theory and Theoretical Physics*, 1:43–110, 1967.

18 Raghav Kulkarni. On the power of isolation in planar graphs. Technical Report TR09-024, Electronic Colloquium on Computational Complexity, 2009.

19 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.

20 Andrea Lapaugh and Christos Papadimitriou. The even-path problem for graphs and digraphs. *Networks Volume 14, Issue 4 , Pages 507 - 513*, 1983.

21 L. Lovász and M.D. Plummer. *Matching Theory*, volume 29. North-Holland Publishing Co, 1986.

22 Meena Mahajan and Kasturi R. Varadarajan. A new NC-algorithm for finding a perfect matching in bipartite planar and small genus graphs (extended abstract). In *STOC*, pages 351–357, 2000.

23 Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM J. Comput.*, 24(5):1002–1017, 1995.

24 Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. John Hopkins University Press, 2001.

25 Ketan Mulmuley, Umesh Vazirani, and Vijay Vazirani. Matching is as easy as matrix inversion. *Combinatorica*, 7:105–113, 1987.

26 Zhivko Prodanov Nedev. Finding an even simple path in a directed planar graph. *SIAM Journal on Computing, Volume 29 , Issue 2, Oct 99, 685-695*, 1999.

27 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.

28 Róbert Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Inf.*, 26(3):279–284, 1988.

29 Raghunath Tewari and N. V. Vinodchandran. Green's theorem and isolation in planar graphs. Technical Report TR10-151, Electronic Colloquium on Computational Complexity, 2010.

30 Leslie G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

31 Vijay Vazirani. NC algorithms for computing the number of perfect matchings in $k_{3,3}$–free graphs and related problems. In *Proceedings of SWAT '88*, pages 233–242, 1988.

32 Heribert Vollmer. *Introduction to Circuit Complexity - A Uniform Approach*. Springer-Verlag, 1999.

Improved Spectral Sparsification and Numerical Algorithms for SDD Matrices

Ioannis Koutis[1], Alex Levin[2], and Richard Peng[3]

1 **Computer Science Department, University of Puerto Rico, Río Piedras**
ioannis.koutis@upr.edu
2 **Department of Mathematics, Massachusetts Institute of Technology**
levin@mit.edu
3 **School of Computer Science, Carnegie Mellon University**
yangp@cs.cmu.edu

Abstract

We present three spectral sparsification algorithms that, on input a graph G with n vertices and m edges, return a graph H with n vertices and $O(n \log n/\epsilon^2)$ edges that provides a strong approximation of G. Namely, for all vectors x and any $\epsilon > 0$, we have

$$(1-\epsilon)x^T L_G x \leq x^T L_H x \leq (1+\epsilon)x^T L_G x,$$

where L_G and L_H are the Laplacians of the two graphs. The first algorithm is a simple modification of the fastest known algorithm and runs in $\tilde{O}(m \log^2 n)$ time, an $O(\log n)$ factor faster than before. The second algorithm runs in $\tilde{O}(m \log n)$ time and generates a sparsifier with $\tilde{O}(n \log^3 n)$ edges. The third algorithm applies to graphs where $m > n \log^5 n$ and runs in $\tilde{O}(m \log_{m/n \log^5 n} n)$ time. In the range where $m > n^{1+r}$ for some constant r this becomes $\tilde{O}(m)$. The improved sparsification algorithms are employed to accelerate linear system solvers and algorithms for computing fundamental eigenvectors of dense SDD matrices.

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 Spectral sparsification, linear system solving

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.266

1 Introduction

The efficient transformation of dense instances of graph problems to nearly equivalent sparse instances is a very powerful tool in algorithm design. The idea, widely known as *graph sparsification*, was originally introduced by Benczúr and Karger [3] in the context of cut problems. Spielman and Teng [10] generalized the *cut-preserving sparsifiers* of Benczúr and Karger to the more powerful *spectral sparsifiers*, which preserve in an algebraic sense the Laplacian matrix of the dense graph. The main motivation of spectral sparsifiers was the design of nearly-linear time algorithms for the solution of symmetric diagonally dominant (SDD) linear systems.[1]

Given that even the existence of cut-preserving sparsifiers is not immediately clear, the result of Benczúr and Karger was indeed very surprising; they proved that, for arbitrary ϵ, cuts can be preserved within a factor of $1 \pm \epsilon$ by a graph with $O(n \log n/\epsilon^2)$ edges. This

[1] A matrix A is SDD if for all i, $A_{ii} \geq \sum_{j \neq i} |A_{ij}|$.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 266–277
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

graph can be computed by a randomized algorithm that runs in $O(m \log^3 n)$ time, where m is the number of edges in the dense graph. Spielman and Teng gave the first construction of spectral sparsifiers, but the edge count of these objects was several log factors bigger than that of Benczúr and Karger's cut-preserving sparsifiers. However, recent progress that we review below allows for the construction of spectral sparsifiers with $O(n \log n/\epsilon^2)$ edges in $\tilde{O}(m \log^3 n \log(1/\epsilon))$ time.[2]

Sparsification can be employed to immediately accelerate algorithms for numerous problems. In several cases and depending on the density of the instance, the sparsification routine dominates the running time of the sparsifier-enhanced algorithm. This provides a strong incentive for speeding up the construction of sparsifiers even further.

This problem was recently undertaken in the context of cut-preserving sparsifiers by Fung *et al.* [5]. Improving upon the work of Benczúr and Karger, they proved that there is an $O(m \log^2 n)$ time algorithm that computes a sparsifier with $O(n \log n/\epsilon^2)$ edges. This stands as the fastest known algorithm with this sparsity guarantee for general graphs. However, Fung *et al.* also showed that we can do even better on slightly more dense graphs. More concretely, they proved that there is an $O(m + n \log n)$ time algorithm that computes a sparsifier with $O(n \log^2 n/\epsilon^2)$ edges. Note that by transitivity, a combination of the two algorithms can produce a graph with $O(n \log n/\epsilon^2)$ edges in $O(m + n \log^4 n/\epsilon^2)$ time. In other words, there is a linear time sparsification algorithm for graphs with more than $n \log^4 n$ edges.

This leads us to the main question we address in this paper: Is something analogous to the result of Fung *et al.* [5] possible for spectral sparsification? We answer the question in the affirmative. We first show that a slight modification of the known algorithm can improve the run time to $\tilde{O}(m \log^2 n \log(1/\epsilon))$. This nearly matches the most general result of [5]. We present two additional sparsification algorithms. The first generates a sparsifier with $\tilde{O}(n \log^3 n/\epsilon^2)$ edges in $\tilde{O}(m \log n)$ time. The second produces a sparsifier with $\tilde{O}(m/\log^2 n)$ edges in $O(m \log_{m/(n \log^5 n)} n)$ time. As in the cut-preserving case, transitivity then allows us to re-sparsify these sparsifiers with the fastest general-case algorithm in order to get a sparsifier with $O(n \log n/\epsilon^2)$ edges.

Applications in numerical algorithms. Sparsification can be used to accelerate the computation of an approximate Fiedler eigenvector of a (normalized) graph Laplacian [11], and more generally of the first non-trivial eigenvector of an SDD matrix L. The approximate eigenvector is a normalized vector x such that $x^T Lx$ is within $1 \pm \epsilon$ of the eigenvalue λ_2. More concretely, by applying the simple inverse power method analyzed in [11] to the sparsifier with $O(n \log^3 n/\epsilon^2)$ edges one can obtain a $1 \pm \epsilon$ approximation of its eigenvector in $O(n \log^5 n \log(1/\epsilon)/\epsilon^2)$ time. However, sparsification preserves the eigenvalues within $1 \pm \epsilon$ and so the computed approximation is a $1 \pm 3\epsilon$ approximation for the dense graph. This implies overall that the Fiedler eigenvector of a graph with $m > n \log^3 n$ can be computed in $O(m \log n + n \log^5 n \log(1/\epsilon)/\epsilon^2)$ time. The previously fastest known algorithm runs in time $O(m \log^2 n \log(1/\epsilon))$. We note here that one practical application of eigenvectors is in partitioning algorithms; the analysis of Cheeger's inequality [4] tells us how to turn an approximate Fiedler vector into a partition. Hence, we give an improvement to the running time of a fundamental graph partitioning algorithm. Finally we note that the computation of additional eigenvectors can be performed in the same amount of time (per vector) by restricting the action of the matrix to the complement of the subspace spanned by the previously computed eigenvectors.

[2] We use the $\tilde{O}()$ notation to hide $\log \log n$ factors.

In addition, the $1 \pm \epsilon$ sparsifiers we obtain can be applied in a standard way as preconditioners for SDD linear systems, giving us a faster solver for these systems. In particular, for the case when m, the number of non-zero entries in the matrix of the system, is greater than n^{1+r} for some small constant r, we can show that our solver approximates a solution with relative error δ in time $\tilde{O}(m \log(1/\delta))$. The previously best known algorithm [8] runs in time $\tilde{O}(m \log n \log(1/\delta))$.

2 Overview of our techniques

2.1 Brief background on spectral sparsification

The first algorithm for edge-efficient spectral sparsifiers was given by Spielman and Srivastava [9]. Their algorithm produces a sparsifier with $O(n \log n/\epsilon^2)$ edges in a very elegant way: it samples edges with replacement. The probability of sampling an edge is proportional to its weight multiplied by its effective resistance in the resistive electrical network associated with the given graph. Computing the effective resistance of a given edge requires—almost by definition—the solution of a linear system on the graph Laplacian.[3] However, Spielman and Srivastava also provided a way of estimating *all* m effective resistances by solving $O(\log n)$ SDD systems; their approach involves characterizing the effective resistances as the squared lengths of vectors and then applying the Johnson-Lindenstrauss (JL) theorem [2]. This holds under the assumption that the SDD solver is direct, i.e. it outputs an exact solution. The use of a nearly-linear time *iterative* solver that computes approximate solutions introduces an additional source of imprecision; Spielman and Srivastava showed that solving the systems up to an inverse polynomial precision is sufficient for sparsification. This brings the running time of their algorithm to $\tilde{O}(m \log^{c+2} n)$, where c is the constant appearing in the running time of the SDD solver.

2.2 The $\tilde{O}(m \log^2 n)$ algorithm

While the work of Spielman and Srivastava did not improve the running time of the SDD solver, it proved to be a decisive step towards the fast SDD solver of Koutis, Miller, and Peng [7, 8], which runs in time $\tilde{O}(m \log n \log(1/\delta))$. Using this solver in the Spielman and Srivastava sparsification sampling scheme immediately yields an $\tilde{O}(m \log^3 n/\epsilon^2)$ time algorithm. This brings us to the first contribution of this paper, a tighter analysis of the Spielman and Srivastava algorithm. In Section 5 we show that a fixed precision from the SDD solver is actually sufficient for sparsification. This decreases the running time to $\tilde{O}(m \log^2 n/\epsilon^2)$. This improvement is included in all our subsequent algorithms.

2.3 The $\tilde{O}(m \log n)$ algorithm

In order to speed up the algorithm further, we need to break the central bottleneck, which comes from having to solve $O(\log n)$ linear systems each of which takes $\tilde{O}(m \log n)$ time. We improve the running time of this step by allowing for cruder, but more easily-computable, approximations of the effective resistances. It was shown in [7] that if we estimate the effective resistances, the Spielman-Srivastava scheme still goes through, but we may need to sample more edges to compensate for the loss of accuracy.

[3] Laplacian matrices are SDD.

In particular, we estimate the effective resistances by using a *spine-heavy* approximation to G. This is a graph that has an extremely good low stretch spanning tree. In [8] it was shown that linear equations in Laplacians of spine-heavy graphs can be solved in $\tilde{O}(m \log(1/\delta))$ time. Further any graph can be easily transformed into a spine-heavy approximation while distorting the effective resistances by at most an $\tilde{O}(\log^2 n)$ factor. Using this spine-heavy approximation in order to quickly estimate effective resistances, and then sampling with respect to these estimates, allows us to get a sparsifier with $O(n \log^3 n/\epsilon^2)$ edges in $\tilde{O}(m \log n)$ time. The details are given in Section 5.

2.4 The $\tilde{O}(m)$ algorithm

Several more obstacles needs to be circumvented for an even faster algorithm. Even assuming a computationally free SDD solver, estimating the effective resistances via the Johnson-Lindenstrauss projection requires operating on m vectors of dimension $O(\log n)$, which is too expensive. This forces us to try to decrease (hopefully down to a constant) the dimension of the projections. Of course this introduces higher distortions in the estimates for the effective resistances, but as we noted above the algorithm can compensate by taking more samples. The second key to our result comes into play here: transitivity. We observe that it is enough to produce a sparsifier with $m' = O(m/\log^2 n)$ edges since we can then run our slightly slower algorithm in time $\tilde{O}(m' \log^2 n) = \tilde{O}(m)$ and get the final sparsifier. This trick allows us to reduce the dimension of the JL projection to a constant, for large enough m. The details are given in Section 6.

However to get these severely distorted estimates for the effective resistances, it is not enough to just take our $O(m \log^2 n)$ algorithm and replace the JL projection by a constant-dimensional one. The remaining bottleneck is the running time of the solver; its construction requires at the minimum the computation of a low-stretch tree which takes $\tilde{O}(m \log n)$ time [1]. The solver steps after the construction of the low-stretch tree take $\tilde{O}(m)$ time on a spine-heavy graph. This implies that we would be able to sparsify in $\tilde{O}(m)$ time if the computation of the low-stretch tree were not an issue.

To solve this problem, we show that every graph can be decomposed into graphs of diameter $O(\log n)$ with relatively few edges between the pieces. Spanning trees with $O(\log n)$ average stretch can be easily computed for each of these pieces, and thus we sparsify them separately and then put the results together. The details are given in Section 7.

3 Background on spectral graph theory

3.1 The graph Laplacian and its pseudoinverse

Let $G = (V, E, w)$ be an undirected weighted graph on n vertices, which we identify with the integers $\{1, 2, \ldots, n\}$, and m edges, where the weight of edge e is given by w_e. The Laplacian of G is denoted by L_G. It is a symmetric $n \times n$ matrix with zero row and column sums, where the (i, j) off-diagonal entry is given by $-w_{(i,j)}$ if (i, j) is an edge of G and 0 otherwise. The ith diagonal entry is given by the weighted degree of vertex i.

If G is a connected graph, then L_G is a matrix of rank $n - 1$, with its kernel spanned by $\mathbf{1}$ (the vector of all 1's). We let L_G^+ denote the Moore-Penrose pseudoinverse of L_G; this is a matrix that acts as the inverse of L_G on $(\ker L_G)^\perp$, and satisfies $L_G^+ L_G = L_G L_G^+ = I_{n-1}$, where I_{n-1} is the projection onto the $(n - 1)$-dimensional image of L_G.

Given the one-to-one correspondence of graphs and their Laplacians we will often apply algebraic notation to graphs, with the obvious meaning.

3.2 Spectral approximation and sparsification

In this paper we concentrate on symmetric diagonally dominant matrices. For two matrices A and B of the same dimension, we write $A \preceq B$ if $x^T Ax \leq x^T Bx$ for all vectors x. For two graphs G and H, we write $G \preceq H$ if the Laplacians satisfy $L_G \preceq L_H$.

▶ **Definition 1.** We say that a graph H is a κ-approximation of a graph G if $G \preceq H \preceq \kappa G$.

It is not hard to show that if H is a graph that κ-approximates a graph G then we have

$$\frac{1}{\kappa} L_G^+ \preceq L_H^+ \preceq L_G^+ \tag{1}$$

▶ **Definition 2.** Given a graph G, we say that a (sparser) graph H is a $1 \pm \epsilon$ **spectral sparsifier** of G if

$$(1-\epsilon)G \preceq H \preceq (1+\epsilon)G. \tag{2}$$

It is easy to see that if H is a $1 \pm \epsilon$ spectral sparsifier of G then $\frac{1}{1-\epsilon} H$ is a graph that $\frac{1+\epsilon}{1-\epsilon}$-approximates G. By the definition, it is also easy to verify **transitivity**. If G_1 is a $1 \pm \epsilon_1$ sparsifier of G and G_2 is a $1 \pm \epsilon_2$ of G_1 then G_2 is a $(1 \pm \epsilon_1)(1 \pm \epsilon_2)$ sparsifier of G.

3.3 Graphs as resistive electrical networks

We can consider our graph G as an electrical network of nodes (vertices) and wires (edges), where edge e has resistivity of w_e^{-1} Ohms.

In this context it is very useful to give another definition of the Laplacian L_G, in terms of its incidence matrix B_G. To define B_G, fix an arbitrary orientation for each edge in G. For a vertex i let χ_i be its $(n \times 1)$ characteristic vector, with a 1 at the ith entry and 0's everywhere else. Let $e = (i, j)$ be an edge and define $b_e = \chi_i - \chi_j$. Then B_G is the $m \times n$ matrix whose eth row is the vector b_e. Let W_G be the $m \times m$ diagonal matrix whose eth diagonal entry is w_e. With these definitions, it is easy to verify that

$$L_G = B_G^T W_G B_G = \sum_{e \in G} w_e b_e b_e^T.$$

For notational convenience, we will drop the subscripts on L_G, B_G, and W_G when the graph we are dealing with is clear from context.

Going back to the electrical analogy, the *effective resistance* between vertices i and j, denoted by $R^G(i, j)$ or $R^G(e)$ when (i, j) is an edge e, is the voltage difference that has to be applied between i and j in order to drive one unit of external current between the two vertices. Algebraically it is given by

$$R^G(i,j) = (\chi_i - \chi_j)^T L_G^+ (\chi_i - \chi_j) \tag{3}$$

The above equation allows us to apply (1) and see that

$$G \preceq H \preceq \kappa G \Rightarrow (1/\kappa) R^G(e) \leq R^H(e) \leq R^G(e). \tag{4}$$

The definition of the effective resistance for (i, j) in (3) shows directly that it can be computed by solving the system $L_G x = (\chi_i - \chi_j)$. In light of this, (4) will be of *central importance* in our proofs. Informally, it states that if H is a κ-approximation of G, then the effective resistance of any edge in G can be approximated by the effective resistance of the same edge in H, which can be done by solving the system $L_H x = (\chi_i - \chi_j)$. This will allow us to construct special approximations H for which solving with L_H is easier than with L_G.

3.4 Low-stretch trees, spine-heavy graphs and SDD solvers

Let T be a spanning tree of G. For any edge $e = (i, j)$ of G, there is a unique path $e_1, e_2, \ldots, e_\nu$ between i and j along edges of T. We say that the *stretch of e in T* is $\text{stretch}_T(e) := w_e \sum_{i=1}^{\nu} w_{e_i}^{-1}$, i.e. the weight of e multiplied by the sum of inverse weights of tree edges on the path from i to j. We denote by $\text{stretch}_T(G)$ the sum of stretches in T of all edges of G, i.e. $\text{stretch}_T(G) = \sum_{e \in G} \text{stretch}_T(e)$.

It is known that every graph G has a spanning tree T with $\text{stretch}_T(G) = \tilde{O}(m \log n)$, known as a **low-stretch tree**. The tree can computed in time $\tilde{O}(m \log n)$ [1, 8]. We call a graph **spine-heavy** if it has a spanning tree with $\text{stretch}_T(G) = O(m/\log n)$. Given a graph G we can compute a spine-heavy graph H that $\tilde{O}(\log^2 n)$-approximates it by computing a low-stretch tree and then scaling up the weights of tree edges in G by the $\tilde{O}(\log^2 n)$ factor. This is summarized in the following lemma.

▶ **Lemma 3.** *Every graph G with n vertices is $\tilde{O}(\log^2 n)$-approximated by a spine-heavy graph H. The graph H can be constructed in time dominated by the computation of a low-stretch tree for G.*

Finally we state a lemma that summarizes the recent work on fast SDD solvers [8].

▶ **Lemma 4.** *Let A be an SDD matrix. There is a symmetric operator $\tilde{A}_\delta$ such that*

$$(1 - \delta)A \preceq \tilde{A}_\delta \preceq (1 + \delta)A$$

and that for any vector b, the vector $\tilde{A}_\delta^+ b$ can be evaluated in $\tilde{O}(m \log n \log(1/\delta))$ time. Moreover, if A is the Laplacian of a spine-heavy graph and its low-stretch tree is given, then $\tilde{A}_\delta^+ b$ can be evaluated in $\tilde{O}(m \log(1/\delta))$ time.

4 The Spielman-Srivastava sampling scheme

Spielman and Srivastava [9] give the following simple algorithm for producing a $1 \pm \epsilon$ sparsifier of a graph G: For each i from 1 to $N = O(n \log n/\epsilon^2)$, we sample an edge e of G from the probability distribution $\boldsymbol{p}$ assigning e a probability p_e proportional to $q_e = w_e R^G(e)$. If we select edge e, we add it to the sparsifier with weight $w_e/(Np_e)$.

This scheme produces a $1 \pm \epsilon$ sparsifier with high probability. An analysis is given in [9], and a different perspective can be found in Srivastava's dissertation [13].

For the efficient implementation of their algorithm Spielman and Srivastava first obtain a different expression for the effective resistance, via a simple algebraic manipulation:

$$\begin{aligned} R^G(i,j) &= (\chi_i - \chi_j)^T L^+ (\chi_i - \chi_j) \\ &= (\chi_i - \chi_j)^T L^+ L L^+ (\chi_i - \chi_j) \\ &= (\chi_i - \chi_j)^T L^+ B^T W^{1/2} W^{1/2} B L^+ (\chi_i - \chi_j) \\ &= \|W^{1/2} B L^+ (\chi_i - \chi_j)\|^2 \end{aligned}$$

The advantage of this definition is that it expresses the effective resistance as the squared Euclidean distance of two points, given by the ith and jth column of the matrix $W^{1/2} B L^+$.

This new expression still involves the solution of a linear system with L. The natural idea is to replace L with an approximation $\tilde{L}$ satisfying the properties described in Lemma 4. So instead of $R^G(i,j)$ we compute the quantities $\hat{R}^G(i,j) = \|W^{1/2} B \tilde{L}_\delta^+ (\chi_i - \chi_j)\|^2$.

Of course, there are still m systems to be solved. To work around this hurdle, Spielman and Srivastava observe that projecting the vectors to an $O(\log n)$-dimensional space preserves

the Euclidean distances within a factor of $1 \pm \epsilon/8$, by the Johnson- Lindenstrauss theorem. Algebraically this amounts to computing the quantities $\|QW^{1/2}B\tilde{L}_\delta^+(\chi_i - \chi_j)\|^2$, where Q is a properly defined random matrix of dimension $k \times m$ for $k = O(\log n)$. The authors invoke the result of Achlioptas [2], which states that one can use a matrix Q each of whose entries is randomly chosen in $\{\pm 1/\sqrt{k}\}$.

The construction of the sparsifiers can can thus be broken up into three steps.

1. Compute $QW^{1/2}B$. This takes time $O(km)$, since B has only two non-zero entries per row.
2. Apply the linear operator $\tilde{L}_\delta^+$ to the k columns of the matrix $(QW^{1/2}B)^T$, using Lemma 4. This gives the matrix $Z = QW^{1/2}B\tilde{L}_\delta^+$.
3. Compute all the (approximate) effective resistances (time $O(km)$) via the square norm of the differences between columns of the matrix Z. Then sample the edges.

5 The first two sparsification algorithms

5.1 The $\tilde{O}(m \log^2 n)$ algorithm

Spielman and Srivastava prove that the approximations $\hat{R}^G(i,j)$ can be used to obtain the sparsifier if they satisfy

$$(1 - \epsilon/4)R^G(i,j) \leq \hat{R}^G(i,j) \leq (1 + \epsilon/4)R^G(i,j).$$

Then they show that this can be satisfied if δ, the accuracy guarantee of the linear system solver, is taken to be an **inverse polynomial** in n. Thus their algorithm is dominated by the second step (the applications of $\tilde{L}_\delta^+$) and takes time $\tilde{O}(m \log^3 n \log(1/\epsilon))$.

The following lemma shows that in fact it is enough to take take δ to be a constant. Furthermore, our proof significantly simplifies the corresponding analysis of [9].

▶ **Lemma 5.** *For a given ϵ, if $\tilde{L}$ satisfies $(1-\delta)L \preceq \tilde{L} \preceq (1+\delta)L$ where $\delta = \epsilon/8$, then the approximate effective resistance values $\hat{R}^G(u,v) = \|W^{1/2}B\tilde{L}^+(\chi_u - \chi_v)\|^2$ satisfy:*

$$(1-\epsilon)R^G(u,v) \leq \hat{R}^G(u,v) \leq (1+\epsilon)R^G(u,v).$$

Proof. We only show the first half of the inequality, as the other half follows similarly. Since L and $\tilde{L}$ have the same null space, by (1) the given condition is equivalent to:

$$\frac{1}{1+\delta}L^+ \preceq \tilde{L}^+ \preceq \frac{1}{1-\delta}L.$$

Since $\frac{1}{1+\delta}L^+ \preceq \tilde{L}^+$, we have

$$\begin{aligned} R^G(u,v) &= (\chi_u - \chi_v)^T L^+ (\chi_u - \chi_v) \\ &\leq (1+\delta)(\chi_u - \chi_v)^T \tilde{L}^+ (\chi_u - \chi_v) \\ &= (1+\delta)(\chi_u - \chi_v)^T \tilde{L}^+ \tilde{L} \tilde{L}^+ (\chi_u - \chi_v). \end{aligned}$$

Applying the fact that $\tilde{L} \preceq (1+\delta)L$ to the vector $\tilde{L}^+(\chi_u - \chi_v)$ in turn gives:

$$\begin{aligned} R^G(u,v) &\leq (1+\delta)^2(\chi_u - \chi_v)^T \tilde{L}^+ L \tilde{L}^+ (\chi_u - \chi_v) \\ &= (1+\delta)^2 \|W^{1/2}B\tilde{L}^+(\chi_u - \chi_v)\|^2 = \hat{R}^G(u,v) \end{aligned}$$

The rest of the proof follows from $\frac{1}{(1+\delta)^2} \leq 1 - \epsilon/4$ by choice of δ. ◀

This proves our first theorem.

▶ **Theorem 6.** *There is a $1 \pm \epsilon$ sparsification algorithm that runs in time $\tilde{O}(m \log^2 n \log(1/\epsilon))$.*

5.2 The $\tilde{O}(m \log n)$ algorithm

In [7] it was proven that if we use estimates to the effective resistances, rather than the true values, the Spielman-Srivastava scheme still works, but in order to produce the sparsifier we have to compensate by taking more samples. Specifically, for $\alpha > 1$, if the probabilities with which we sample all edges are at least $1/\alpha$ of the true values, then we have to take α times as many samples. This is formalized in the following lemma.

▶ **Lemma 7.** *Suppose that we run the Spielman-Srivastava algorithm and sample edges with probabilities proportional to $\tilde{q}_e$ such that $(1/\alpha)q_e \leq \tilde{q}_e \leq q_e$ for all edges e. Then, taking α times as many samples gets us a $1 \pm \epsilon$ sparsifier with the same high probability guarantee as the Spielman-Srivastava algorithm run with probabilities proportional to q_e.*

We are now ready to state our second theorem.

▶ **Theorem 8.** *There is a $1 \pm \epsilon$ sparsification algorithm for graphs with $m > n \log^3 n$ edges that runs in time $\tilde{O}(m \log n \log(1/\epsilon))$. The output sparsifier contains $\tilde{O}(n \log^3 n/\epsilon^2)$ edges.*

Proof. Given the input graph G we construct a spine-heavy graph H that $\tilde{O}(\log^2 n)$-approximates G. The construction can be done in time $\tilde{O}(m \log n)$, by Lemma 3. We run the Spielman-Srivastava scheme (Section 4) on H to approximate the effective resistances $R^H(i,j)$ within a factor of $1 \pm \epsilon$. Step 2 of the Spielman-Srivastava scheme runs in $\tilde{O}(m \log n \log(1/\epsilon))$ time on H, by Lemma 4. We adjust the approximate effective resistances in H down by a factor of $1 + \epsilon$ to accommodate for the upper side of the error in Lemma 5. Then, by (2) the calculated approximate effective resistances satisfy

$$\frac{1}{\tilde{O}(\log^2 n)} R^G(i,j) \leq \hat{R}^H(i,j) \leq R^G(i,j).$$

Finally we let $\tilde{q}_e = w_e \hat{R}^H(i,j)$ for all edges $e = (i,j)$ of G and sample the edges of G with probabilities proportional to $\tilde{q}_e$. By Lemma 7 we see that we get a $1 \pm \epsilon$ sparsifier with $\tilde{O}(n \log^3 n/\epsilon^2)$ edges. ◀

6 Effective resistances via very-low dimensional projections

With the improvement of the last section, all three steps of the Spielman-Srivastava algorithm take $\tilde{O}(m \log n)$ time; our goal now is to reduce this to $\tilde{O}(m)$. The extra logarithm in the current implementation is due to the dimension $k = O(\log n)$ of the projection matrix Q, and we address this issue here.

It is worth noting that once we have a sparsifier H with $O(\frac{m}{\log^2 n})$ edges such that

$$\left(1 - \frac{\epsilon}{2}\right) G \preceq H \preceq \left(1 + \frac{\epsilon}{2}\right) G,$$

we can afford to fully $(1 \pm \frac{\epsilon}{2})$-sparsify that H using our $\tilde{O}(m \log^2 n)$ algorithm. The sparsifier of H (with $O(n \log n/\epsilon^2)$ edges) will then be a $1 \pm \epsilon$-sparsifier for G.

Since we can take more samples, we are able to underestimate probabilities more aggressively by decreasing the dimension we project onto, and still get a good approximation to G with high probability. In order to show that we do not underestimate effective resistances by too much, we need a more detailed understanding of the relationship between the dimension k and the approximation guarantee. This is provided by the version of the Johnson-Lindenstrauss theorem stated as Lemma 7 of [6]:

▶ **Lemma 9.** *Let u be a unit vector in $\mathbb{R}^\nu$. For any given positive integers k, let $U_1, \ldots, U_k$ be random vectors chosen independently from the ν-dimensional Gaussian distribution $N^\nu(0,1)$. For $X_i = u^T U_i$, define $W = W(u) = (X_1, \ldots, X_k)$ and $L = L(u) = \|W\|^2$. Then for any $\beta > 1$:*

1. $E(L) = k$,
2. $\Pr[L \geq \beta k] < O(k)\exp(-\frac{k}{2}(\beta - (1 + \ln \beta))$,
3. $\Pr[L \leq k/\beta] < O(k)\exp(-\frac{k}{2}(\beta^{-1} - (1 - \ln \beta))$.

Following standard analysis of the Johnson-Lindenstrauss theorem, we see that this lemma essentially gives us the probability of increasing or decreasing sizes of a given vector by a certain factor when we multiply the vector by a random matrix of Gaussian entries.[4] Roughly, the third part states that for a given small constant $r \ll 1$, the probability of underestimating distances (and hence effective resistances in our application) by an n^r factor is around $O(n^{-rk/2})$. By setting k sufficiently large and applying a union bound, we obtain that with high probability all estimates are at least $\Omega(n^{-r})$ of the true quantities required by the Spielman-Srivastava algorithm.

Combining this with the fact that weight times effective resistance is upper bounded by 1, one can show by concentration of measure theorems that the normalizing factor (i.e. the weighted sum of the estimated effective resistances) stays within a constant factor of its true value with high probability. Therefore, with high probability we underestimate the edge selection probabilities by at most a factor of $O(n^r)$. The number of samples we need to take as a result is $n^{1+r}\log n$. As long as this is smaller than $m/\log^2 n$ we can sparsify in $\tilde{O}(m)$ time. This shows that as long as m is big enough relative to n, we can sparsify in linear time, as we claimed in the introduction. We formalize this argument below.

▶ **Lemma 10.** *There is an algorithm that, on input a graph G with n vertices, m edges, a low-stretch spanning tree for G with total stretch $\tilde{O}(m \log n)$, and a parameter t, generates a $1 \pm \epsilon$ sparsifier with $\tilde{O}(\frac{m}{t}\log n/\epsilon^2)$ edges in $\tilde{O}(m \log_{\frac{m}{3t\log^2 n}} n \log(1/\epsilon))$ time.*

Proof. We first construct in $O(m)$ time the spine-heavy graph G' that $\tilde{O}(\log^2 n)$-approximates G. We then apply the Spielman-Srivastava sampling scheme in order to estimate the effective resistances in G'.

Invoking Part 3 of Lemma 9 with $\beta = \frac{m}{nt\log^2 n}$ shows us that when we project onto k dimensions, the probability of underestimating by a factor of β is at most:

$$O(k)\exp\left(\frac{k}{2}(1 - \beta^{-1} - \ln\beta)\right) \leq O(k)\exp\left(\frac{k}{2}(1 - \ln\beta)\right) \leq O(k)(3/\beta)^{\frac{k}{2}}$$

where the first inequality follows from $k/2 \geq 0$ and $1 - \beta^{-1} \leq 1$. So when $(3/\beta)^{\frac{k}{2}} = n^{-d}$, taking a union bound over all $m \leq n^2$ edges gives that no edge's effective resistance is underestimated by more than a factor of β. The requirement on k imposed by this is:

$$\begin{aligned} O(k)(3/\beta)^{\frac{k}{2}} &\leq n^{-d} \\ k &\geq 2d\log_{\beta/3} n + \log_{\beta/3} k + O(1) \end{aligned}$$

Setting d to be some constant and taking the value of β as before we see that taking $k = O(\log_{\frac{m}{3nt\log^2 n}} n)$ will give us the required high probability claims.

[4] This is a minor difference from previous parts, where we use matrices entries randomly chosen in $\pm 1/\sqrt{k}$

This shows that projecting in order to estimate effective resistances and using these to estimate edge selection probabilities will give us values that are at least an $nt \log^2 n/m$ factor of the true value (for β as above). Following the proof of Lemma 5 we can see that using an approximate solver introduces a small multiplicative error. Using the fact that G' is a graph that $O(\log^2 n)$-approximates G, we see that this method produces approximates probabilities in G that are at least a factor of $\frac{nt}{m}$ of the true values.

Consider sampling with these estimated probabilities. Then, by the discussion at the beginning of Section 5.2 with $\alpha = m/(nt)$, we see that to sparsify we need to take $O(\frac{m}{t} \log n\epsilon^{-2})$ samples.

The running time of this process is dominated by amount of time it takes to do k solves in $L_{G'}$, namely $O(km \log(1/\epsilon))$ by Lemma 4. For the choice of k as before this is $\tilde{O}(m \log_{\frac{m}{3t \log^2 n}} n \log(1/\epsilon))$, as required. ◀

▶ **Theorem 11.** *Given a graph G with n vertices, m edges such that $m > n \log^5 n$, and a low-stretch spanning tree with stretch $\tilde{O}(m \log n)$, we can generate a $1 \pm \epsilon$-sparsifier H of G with $O(n \log n/\epsilon^2)$ edges in $\tilde{O}(m \log_{\frac{m}{n \log^5 n}} n \log(1/\epsilon))$ time.*

Proof. Applying Lemma 10 with $t = O(\log^3 n)$ gives a graph with $\tilde{O}(\frac{m}{\log^2 n})$ edges that is a $1 \pm \epsilon$-sparsifier. This graph can in turn be sparsified in $\tilde{O}(\frac{m}{\log^2 n} \log^2 n) = \tilde{O}(m)$ time, by Theorem 8. ◀

7 Improved sparsification via graph decompositions

Theorem 11 reveals that the computation of the low-stretch tree of the input graph is the final bottleneck on our way to getting the faster algorithms. In order to solve this problem, we no longer compute a low-stretch spanning tree for the entire graph. Instead, we decompose the graph into subgraphs for which we can trivially find low-stretch spanning trees and we sparsify each subgraph separately. The decomposition is based on the following simple fact about low diameter graphs:

▶ **Lemma 12.** *Given an unweighted graph with n vertices, m edges, and diameter $O(\log n)$, finding a breadth-first search (BFS) tree in $O(m)$ time gives low stretch spanning tree with average stretch $O(\log n)$.*

We can now apply low diameter decomposition to extend this to arbitrary undirected graphs losing an extra factor of $\log \log n$. The variant of low diameter decomposition that we use can be best described using the following lemma (see, e.g., [14, Lemma 4]).

▶ **Lemma 13.** *Given an undirected, unweighted graph with n vertices and m edges, we can partition it into pieces of $O(\log n)$ diameter so that at most $m/2$ edges are between the pieces.*

Applying this $O(\log \log n)$ times and sparsifying the edges between pieces each time gives the claim for arbitrary unweighted graphs:

▶ **Theorem 14.** *Given an undirected, unweighted graph G with n vertices and m edges such that $m > \Omega(n \log^4 n)$, we can output a sparsifier H with $\tilde{O}(n \log n/\epsilon^2)$ edges in $\tilde{O}(m \log_{\frac{m}{3n \log^4 n}} n \log(1/\epsilon))$ time.*

Proof. We create $G_1, \ldots G_l$ where $l = 4 \log \log n$ as follows. Given $G_1 \ldots G_i$, we partition $E(G) \setminus E(G_1) \ldots \setminus E(G_i)$ into low diameter pieces using Lemma 13 and let G_{i+1} be edges

with both endpoints in the same piece that's not in some G_j with $j \leq i$. Applying guarantees of Lemma 13 inductively gives $|E(G_i)| \leq 2^{-i}E(G) = 2^{-i}m$, and specifically $|E(G_l)| \leq \frac{m}{\log^2 n}$. Therefore G_l can be sparsified to H_l via the slower algorithm in time $\tilde{O}(m \log(1/\epsilon))$.

We now turn our attention to $G_1 \ldots G_{l-1}$. If G_i contains less than $O(m/(\log^2 n/4 \log\log n))$ edges, it can be left unsparsified. Otherwise, since a low-stretch tree can be obtained trivially, we can sparsify it by means of Lemma 10. Concretely, by letting $t = \log^2 n$ we get graphs $H_1, \ldots, H_{l-1}$ (the $1 \pm \epsilon$-sparsifiers of the corresponding G_i) such that

$$(1-\epsilon)G_i \preceq \hat{H}_i \preceq (1+\epsilon)G_i,$$

in total time $\tilde{O}(m \log_{\frac{m}{n \log^4 n}} n \log(1/\epsilon))$. Letting $\hat{H} = H_l + \sum_{i<l} H_i$ gives a sparsifier with $\tilde{O}(\frac{m}{\log n}/\epsilon^2)$ edges, which can in turn be sparsified in $\tilde{O}(m)$ time to generate H with $O(n \log n/\epsilon^2)$ edges. ◀

For weighted graphs, we partition edges by weights into buckets and sparsify each subgraph. Combining this type of partition with the lemmas above gives a sparsifier with $O(\log n)$ loss in edge count.

▶ **Theorem 15.** *Given a graph G with n vertices, m edges such that $m > \Omega(n \log^5 n)$ we can compute in $\tilde{O}(m \log_{\frac{m}{3n \log^5 n}} n \log(1/\epsilon))$ time a sparsifier for it with $\tilde{O}(n \log n/\epsilon^2)$ edges.*

Proof. By Section 10.2 of [12], edges whose endpoints are connected by a path with weights that are larger by a factor of n^3 can be discarded without significant changes to the spectral structure of the graph. Then grouping the edges by weights into buckets containing edges with weights $[(1+\epsilon)^i W_{\min}, (1+\epsilon)^{i+1} W_{\min}]$ gives a partition of G into $G_1, \ldots, G_l$ such that $|V(G_1)| + \cdots + |V(G_l)| \leq O(n \log n \epsilon^{-1})$. By Lemma 10 with $t = \log^2 n$, each G_i where $|E(G_i)| \geq |V(G_i)|\frac{m}{n \log^3 n}$ can be sparsified to a graph with $|V(G_i)|\frac{m}{n \log^3 n}$ edges in $\tilde{O}(\log_{\frac{m}{n}} n(|E(G_i)| + |V(G_i)| \log n))$ time. The total running time of this part is $\tilde{O}(\log_{\frac{m}{n}} n(m + n \log^2 n \epsilon^{-1}))$. Then the total number of edges remaining is at most $\frac{m}{n \log^3 n} \sum_i |V(G_i)| \leq \frac{m}{\log^2 n}$. This graph can in turn be sparsified in $\tilde{O}(m)$ time to give a sparsifier with $O(n \log n/\epsilon^2)$ edges. ◀

8 Final Remarks

We remark that the $\tilde{O}(m)$ sparsification algorithm of this paper relies crucially on graph decompositions. However it seems natural to conjecture that decompositions are not necessary, and that the same upper bound can be obtained via straightforward sampling scheme. We believe that this is an interesting question that would potentially lead to a deeper understanding of low-stretch subgraph computations.

On the other hand the original algorithm of Spielman and Teng remains the only known combinatorial sparsification algorithm that does not rely on solving systems. Designing a spectral sparsification algorithm that does not depend on a linear system solver and that outputs a very sparse graph with $O(n \log n)$ or $O(n \log^2 n)$ edges is a challenging open problem. Given that it may be impossible to achieve this, it also makes sense to ask for algorithms that compute very sparse κ-approximations for small κ. Such algorithms could play a significant role in the development of more practical SDD solvers.

Finally, the possibility of a linear time sparsification algorithm for graphs with $m = O(n \log^c n)$ edges is left open, and we believe it poses an interesting open problem.

Acknowledgments We would like to thank Jonathan Kelner and Gary Miller for useful discussions and the anonymous referees for carefully reading this paper. Ioannis Koutis and Richard Peng are partially supported by the National Science Foundation under grant number CCF-1018463. Richard Peng was at Microsoft Research New England for part of this work and is supported by a Microsoft Research Fellowship. Alex Levin is supported by a National Science Foundation graduate fellowship.

References

1 Ittai Abraham, Yair Bartal, and Ofer Neiman. Nearly tight low stretch spanning trees. *CoRR*, abs/0808.2017, 2008.

2 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, 2001.

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, 1996.

4 Fan Chung. Random walks and local cuts in graphs. *Linear Algebra and its applications*, 423(1):22–32, 2007.

5 Wai Shing Fung, Ramesh Hariharan, Nicholas J. A. Harvey, and Debmalya Panigrahi. A general framework for graph sparsification. In *STOC '11: Proceedings of the 43rd ACM Symposium on Theory of Computing*, pages 71–80, 2011.

6 Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *STOC '98: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613, 1998.

7 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.

8 Ioannis Koutis, Gary L. Miller, and Richard Peng. A nearly-$m \log n$ solver for SDD linear systems. In *FOCS '11: Proceedings of the 52nd Annual IEEE Symposium on Foundations of Computer Science*, 2011.

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, 2008.

10 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 36th Annual ACM Symposium on Theory of Computing*, pages 81–90, 2004.

11 Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems. *CoRR*, abs/cs/0607105, 2006.

12 Daniel A. Spielman and Shang-Hua Teng. Spectral sparsification of graphs. *CoRR*, abs/0808.4134, 2008.

13 Nikhil Srivastava. Spectral sparsification and restricted invertibility, 2010. PhD Thesis, Yale University.

14 Luca Trevisan. Approximation algorithms for unique games. In *FOCS '05: Proceedings of the 46th IEEE Symposium on Foundations of Computer Science*, pages 5–34, 2005.

Linear min-max relation between the treewidth of H-minor-free graphs and its largest grid minor

Ken-ichi Kawarabayashi[*1] **and Yusuke Kobayashi**[†2]

1 **National Institute of Informatics**
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo, Japan
k_keniti@nii.ac.jp

2 **University of Tokyo**
7-3-1, Hongo, Bunkyo-ku, Tokyo, Japan
kobayashi@mist.i.u-tokyo.ac.jp

Abstract

A key theorem in algorithmic graph-minor theory is a min-max relation between the treewidth of a graph and its largest grid minor. This min-max relation is a keystone of the Graph Minor Theory of Robertson and Seymour, which ultimately proves Wagner's Conjecture about the structure of minor-closed graph properties. In 2008, Demaine and Hajiaghayi proved a remarkable linear min-max relation for graphs excluding any fixed minor H: every H-minor-free graph of treewidth at least $c_H\, r$ has an $r \times r$-grid minor for some constant c_H. However, as they pointed out, there is still a major problem left in this theorem. The problem is that their proof heavily depends on Graph Minor Theory, most of which lacks explicit bounds and is believed to have very large bounds. Hence c_H is not explicitly given in the paper and therefore this result is usually not strong enough to derive efficient algorithms.

Motivated by this problem, we give another (relatively short and simple) proof of this result without using big machinery of Graph Minor Theory. Hence we can give an explicit bound for c_H (an exponential function of a polynomial of $|H|$). Furthermore, our result gives a constant $w = 2^{O(r^2 \log r)}$ such that every graph of treewidth at least w has an $r \times r$-grid minor, which improves the previously known best bound $2^{\Theta(r^5)}$ given by Robertson, Seymour, and Thomas in 1994.

1998 ACM Subject Classification G.2.2 Graph Theory

Keywords and phrases grid minor, treewidth, graph minor

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.278

1 Introduction

One of the deepest and most far-reaching theories of the recent 20 years in the realm of discrete mathematics and theoretical computer science is Graph Minor Theory developed by Robertson and Seymour in a series of over 20 papers spanning the last 20 years. The original goal of this work, now achieved, was to prove Wagner's Conjecture [26], which can be stated as follows: every minor-closed graph property (preserved under taking of minors) is characterized by a finite set of forbidden minors. This theorem has a powerful algorithmic consequence:

* Research partly supported by Japan Society for the Promotion of Science, Grant-in-Aid for Scientific Research, by C & C Foundation, by Kayamori Foundation and by Inoue Research Award for Young Scientists.

† Supported by Grant-in-Aid for Scientific Research and by the Global COE Program "The research and training center for new development in mathematics", MEXT, Japan.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 278–289

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

every minor-closed graph property can be decided by a polynomial-time algorithm. This follows from another important result in Graph Minor Theory which gives a polynomial time algorithm to test whether or not a given graph has a fixed graph as a minor. One of the most central concepts, introduced early on, is the notion of *treewidth* [24]. Treewidth has obtained immense attention ever since, especially because many NP-hard problems can be handled efficiently on graphs of bounded treewidth [1]. In fact, all problems that can be defined in monadic second-order logic are solvable for graphs of bounded treewidth [4]. But perhaps even more importantly, Graph Minor Theory gives a powerful and vast toolkit of concepts and ideas to handle graphs and understand their structure. Indeed, a huge body of work has evolved that applies and extends these ideas in various fields of discrete mathematics and computer science.

A keystone in the proof of these theorems, and many other theorems, is a grid-minor theorem [24]: any graph of treewidth at least some $f(r)$ is guaranteed to have the $r \times r$ grid graph as a minor. This gird-minor theorem played a key role for the graph minor algorithm (c.f., the disjoint paths problem [16, 17, 25, 27, 28]). It also played a key role for some other deep applications (e.g., [12, 14, 15, 20]).

Such grid-minor theorems have also played a key role for many algorithmic applications, in particular via the bidimensionality theory (e.g., [5, 6, 7, 9]), including many approximation algorithms, PTASs, and fixed-parameter algorithms. These include feedback vertex set, vertex cover, minimum maximal matching, face cover, a series of vertex-removal parameters, dominating set, edge dominating set, R-dominating set, connected dominating set, connected edge dominating set, connected R-dominating set, and unweighted TSP tour.

The grid-minor theorem of [24] has been extended, improved, and re-proved by Robertson, Seymour, and Thomas [29], Reed [22], and Diestel, Jensen, Gorbunov, and Thomassen [11]. The best bound known for general graphs is superexponential: every graph of treewidth more than 20^{2r^5} has an $r \times r$ grid minor [29]. We note that as a corollary of our main theorem in this paper, we improve this bound in Corollary 2. Robertson et al. [29] conjecture that the bound on $f(r)$ can be improved to a polynomial $r^{\Theta(1)}$; the best known lower bound is $\Omega(r^2 \log r)$.

A linear upper bound has been shown for planar graphs [29] and bounded genus graphs [6]. Recently this min-max relation is also established for graphs excluding any fixed minor H: every H-minor-free graph of treewidth at least $c_H\, r$ has an $r \times r$ grid minor for some constant c_H [8]. This bound leads to many powerful algorithmic results on H-minor-free graphs [3, 8, 9, 13] that are previously not known.

However, as Demaine and Hajiaghayi pointed out in [8] (also see [10]), there are still major problems left in this grid-minor theorem for H-minor-free graphs, in particular in algorithmic graph-minor theory. The biggest problem is how large the constant c_H in the grid-minor theorem for H-minor-free graphs is. In particular, how does it depend on H? This constant is particularly important because it is in the exponent of the running times of many algorithms, as mentioned in [8, 10]. The current results (e.g., [8]) heavily depend on Graph Minor Theory, most of which lacks explicit bounds and is believed to have very large bounds. Recently, there is a simplified proof of Graph Minor Theory [18], but the bound is still huge. For this reason, improving the constants, even for special classes of graphs, and presumably using different approaches from graph minors, is an important theoretical and practical challenge.

Perhaps, Demaine, Hajiaghayi and Kawarabayashi [10] are the first to try to attack this issue, and they gave explicit bounds for the case of $K_{3,k}$-minor-free graphs, an important class of apex-minor-free graphs extending bounded genus graphs. The bounds are not too

small but are a vast improvement over previous bounds (in particular, much smaller than $2 \uparrow |V(H)|$, where $2 \uparrow n$ denotes a tower $2^{2^{2^{\cdot^{\cdot^{\cdot}}}}}$ involving n 2's).

In this paper, we resolve this issue. More precisely, our main theorem is the following.

▶ **Theorem 1.** *For any fixed graph H and for any positive integer r, there exists a constant $w = |V(H)|^{O(|E(H)|)} \cdot r$ satisfying the following. If G does not contain an H-minor but has treewidth is at least w, then G has an $r \times r$-grid minor. Moreover, there is an algorithm, whose running time is a polynomial in $|V(G)|$ and w, to output either a tree-decomposition of width at most w, an $r \times r$-grid minor, or an H-minor in a given graph.*

Let us emphasize that, unlike the algorithms using the graph minor theory [8], no huge function of $|H|$ is involved in the above algorithm.

Furthermore, by setting H as an $r \times r$-grid with r^2 vertices and $2r^2 - 2r$ edges, Theorem 1 implies the following as a corollary, which improves the previously known best bound 20^{2r^5} given in [29] for large r.

▶ **Corollary 2.** *There exists a constant $w = 2^{O(r^2 \log r)}$ such that every graph of treewidth at least w has an $r \times r$-grid minor.*

To the best of our knowledge, Theorem 1 is the only grid-minor theorem with an explicit bound other than for planar graphs [29], bounded-genus graphs [6], and $K_{3,k}$-minor-free graphs [10]. Our theorem also leads to several algorithms with explicit and improved bounds on their running time, as mentioned above, in particular via the bidimensionality theory (e.g., [5, 6, 7, 9]).

In addition, the proof techniques are interesting in their own right, for example, the path-intertwining technique used in many contexts (see, e.g., [2, 19]), together with some techniques from Diestel et al. [11].

This paper is organized as follows. In Section 2, we give notations and results that are needed in this paper. In Section 3, we adapt tools from Diestel et al. [11]. Our key lemmas are provided in Section 4. Finally in Section 5, we give our main proof of Theorem 1.

2 Preliminaries

In this paper, n and m always mean the number of vertices of a given graph and the number of edges of a given graph, respectively. For $X \subseteq V$ in a graph $G = (V, E)$, let $N_G(X)$ denote the set of vertices in $V \setminus X$ that are adjacent to X. For simplicity, for $v \in V$, $N_G(\{v\})$ is denoted by $N_G(v)$. A *separation* (A, B) is that $G = A \cup B$, there are no edges in $E(A) \cap E(B)$, and moreover both $A - B$ and $B - A$ are nonempty. The order of the separation (A, B) is $|V(A) \cap V(B)|$. An $r \times r$ *grid* is a graph which is isomorphic to the graph W_r obtained from Cartesian product of paths of length $r - 1$, with vertex set $V(W_r) = \{(i, j) \mid 1 \leq i \leq r,\ 1 \leq j \leq r\}$ in which two vertices (i, j) and (i', j') are adjacent if and only if $|i - i'| + |j - j'| = 1$.

A *tree decomposition* of a graph G is a pair $(T, \mathcal{W})$, where T is a tree and $\mathcal{W}$ is a family $\{W_t \mid t \in V(T)\}$ of vertex sets $W_t \subseteq V(G)$, such that the following two properties hold:

(1) $\bigcup_{t \in V(T)} W_t = V(G)$, and every edge of G has both ends in some W_t.
(2) If $t, t', t'' \in V(T)$ and t' lies on the path in T between t and t'', then $W_t \cap W_{t''} \subseteq W_{t'}$.

The *width* of a tree decomposition $(T, \mathcal{W})$ is $\max_{t \in V(T)} |W_t| - 1$. The *treewidth* of a graph G is the minimum width over all possible tree decompositions of G.

A *linkage* $\mathcal{P}$ is a set of mutually vertex-disjoint paths in a graph. For two vertex sets Z_1 and Z_2, $\mathcal{P}$ is a Z_1-Z_2 *linkage* if each member is a path from Z_1 to Z_2. The *order* of the linkage, denoted by $|\mathcal{P}|$ is the number of paths. In slightly sloppy notation, sometimes we will identify a linkage $\mathcal{P}$ with the subgraph consisting of the paths in $\mathcal{P}$. For a linkage $\mathcal{P} = \{P_1, \ldots, P_p\}$ in G, a $\mathcal{P}$-*bridge* in G is either an edge $e \in E(G) \setminus E(\mathcal{P})$ whose endpoints are both in $\mathcal{P}$, or a subgraph of G consisting of a connected component C of $G - \mathcal{P}$ together with all edges joining C and $\mathcal{P}$. The vertices of a $\mathcal{P}$-bridge B in $\mathcal{P} \cap B$ are called *attachments* of B, and we say that B is *attached* to $\mathcal{P}$ at these vertices. Given any two subpaths P and Q contained in the linkage $\mathcal{P}$, we say that they are *adjacent* if there exists a $\mathcal{P}$-bridge which intersects with both P and Q.

Now, we present some known results on mesh and treewidth, which will be used in the next section. For an integer α, we call a set $X \subseteq V(G)$ α-*connected* in G if $|X| \geq \alpha$ and for all subsets $Y, Z \subseteq X$ with $|Y| = |Z| \leq \alpha$, there are $|Y|$ mutually vertex-disjoint paths in G from Y to Z. Note that the sets Y and Z are not required to be disjoint. If $X = V(G)$, then we say G is α-connected. An α-connected set X is *externally* α-*connected* if, in addition, the required paths do not contain any vertex in X except their endpoints. Following [11], let us call a separation (A, B) a *premesh* if all the edges with both end vertices in $V(A) \cap V(B)$ lie in A, and A contains a tree T with the following properties:

1. T has maximum degree at most three;
2. every vertex of $A \cap B$ lies in T and has degree at most two in T; and
3. T has a leaf in $A \cap B$.

A premesh (A, B) is called an α-*mesh* if $V(A \cap B)$ is externally α-connected in B, and the graph $G = A \cup B$ is said to *have* this premesh or α-mesh.

Among useful lemmas on the α-mesh, Diestel et al. [11] proved the following lemmas.

▶ **Lemma 3.** *Let G be a graph and let $\beta \geq \alpha \geq 1$ be integers. If G has no α-mesh of order β, then G has treewidth $< \alpha + \beta - 1$.*

▶ **Lemma 4.** *Let $\beta \geq 2$ be an integer. Let T be a tree of maximum degree ≤ 3 and $X \subseteq V(T)$ be a vertex set with $|X| \geq \beta$. Then T has an edge set $F \subseteq E(G)$ such that every component of $T - F$ has at least β vertices and at most $2\beta - 2$ vertices in X, except that one such component may have fewer vertices in X.*

3 Finding good linkages

In this section, we show that graphs with large treewidth have a pair of linkages with some good properties. Such linkages will be used to construct a grid-minor or an H-minor in Sections 4 and 5. The following lemma is obtained from the arguments in [11], but we describe the proof for completeness.

▶ **Lemma 5.** *For a graph H with h vertices and for integers k, p', there exists an integer $w = (kh)^{O(|E(H)|)} \cdot p'$ satisfying the following. If a graph G has treewidth at least w, then either G contains an H-minor or two linkages $\mathcal{P}$ and $\mathcal{Q}$ such that*

(C1) *$|\mathcal{P}| \geq p'$ and $|\mathcal{Q}| \geq 3k^2|\mathcal{P}|$,*
(C2) *each path in $\mathcal{Q}$ hits all but at most $|\mathcal{P}|/3k^2$ paths in $\mathcal{P}$, and*
(C3) *$\mathcal{P}$ is a Z_1-Z_2 linkage for some $Z_1, Z_2 \subseteq V(G)$ such that for each edge $e \in E(\mathcal{P})$, $(\mathcal{P} \cup \mathcal{Q}) - e$ has no Z_1-Z_2 linkage.*

Proof. Let $c = 3k^2h^2$ and let $\alpha = c^{2|E(H)|-1}p'$. We show that $w = (2h + 2)\alpha$ is a desired integer.

Suppose that G has treewidth at least w. By Lemma 3, there is an α-mesh of order at least $(2h+1)(\alpha-1)$. Let $T \subseteq A$ be a tree associated with the premesh (A, B). Let $X = V(A \cap B) \subseteq V(T)$. By Lemma 4, T has at least h disjoint subtrees each containing at least h vertices of X. Let $A_1, \ldots, A_h$ be the vertex sets of these subtrees. Then by the definition of k-mesh, B contains a set $\mathcal{P}_{ij}$ of k mutually vertex-disjoint paths between A_i and A_j that have no inner vertices in A.

Let us identify the index set $\{0, 1, \ldots, h-1\}$ and the vertex set of H, and let us impose a linear ordering on the index pairs ij by fixing a bijection $f : \{ij \mid 1 \leq i < j \leq h\}$ to $\{0, \ldots, \binom{h}{2} - 1\}$ such that $f(ij) < |E(H)|$ if and only if $ij \in E(H)$. Let $l^* \leq \binom{h}{2}$ be a maximum integer such that for all $0 \leq l < l^*$ and all i, j, there exist sets $\mathcal{P}_{ij}^l$ satisfying the following conditions.

1. $\mathcal{P}_{ij}^l$ is a set of mutually vertex-disjoint paths from A_i to A_j in B that hit A only in their end points.
2. If $f(ij) < l$, then $\mathcal{P}_{ij}^l$ has exactly one path P_{ij}, and P_{ij} does not meet any paths in $\mathcal{P}_{st}^l$ with $ij \neq st$.
3. If $f(ij) = l$, then $|\mathcal{P}_{ij}^l| = \alpha/c^{2l}$.
4. If $f(ij) > l$, then $|\mathcal{P}_{ij}^l| = \alpha/c^{2l+1}$.
5. If $l = f(st) < f(ij)$, then for every edge $e \in E(\mathcal{P}_{ij}^l) \setminus E(\mathcal{P}_{st}^l)$, there are no k/c^{2l+1} vertex-disjoint paths from A_i to A_j in the graph $(\mathcal{P}_{ij}^l \cup \mathcal{P}_{st}^l) - e$.

If $l^* \geq |E(H)|$, then we are done since there is an H-minor. Hence we may assume that $l^* < |E(H)|$.

We shall first prove that $l^* > 0$. Let $st = f^{-1}(0)$ and put $\mathcal{P}_{st}^0 := \mathcal{P}_{st}$. For any ij with $f(ij) > 0$, let $F_{ij} \subseteq E(\mathcal{P}_{ij}) \setminus E(\mathcal{P}_{st}^0)$ be a maximal edge set such that there are still α/c vertex-disjoint paths from A_i to A_j in $(\mathcal{P}_{ij} \cup \mathcal{P}_{st}^0) - F_{ij}$, and define $\mathcal{P}_{ij}^0$ as such a set of paths. Then it is easy to see that $\mathcal{P}_{ij}^0$ satisfies the above conditions. This proves that $l^* > 0$.

Since $l^* > 0$, by the maximality of l^*, the above five conditions are satisfied for $l < l^*$ but cannot be satisfied for $l = l^*$. Let $st = f^{-1}(l^* - 1)$. We claim that there is no path $P \in \mathcal{P}_{st}^{l^*-1}$ such that P avoids a set $\mathcal{L}_{ij}$ of some $|\mathcal{P}_{ij}^{l^*-1}|/c$ paths in $\mathcal{P}_{ij}^{l^*-1}$ for all ij with $f(ij) \geq l^*$. Suppose such a path P exists. Let $s't' := f^{-1}(l^*)$ and define $\mathcal{P}_{s't'}^{l^*} := \mathcal{L}_{s't'}$. Let $\mathcal{P}_{st}^{l^*} := \{P\}$ and $\mathcal{P}_{ij}^{l^*} := \mathcal{P}_{ij}^{l^*-1}$ for $f(ij) < l^* - 1$. For each ij with $f(ij) > l^*$, let $F_{ij} \subseteq E(\mathcal{L}_{ij}) \setminus E(\mathcal{L}_{s't'}^{l^*})$ be a maximal edge set such that there are still $|\mathcal{P}_{ij}^{l^*-1}|/c^2$ vertex-disjoint paths from A_i to A_j in $(\mathcal{L}_{ij} \cup \mathcal{L}_{s't'}^{l^*}) - F_{ij}$ and define $\mathcal{P}_{ij}^{l^*}$ as such a set of paths. Then these would give rise to a family of sets $\mathcal{P}_{ij}^{l^*}$, a contradiction to the maximality of l^*.

Thus for every path $P \in \mathcal{P}_{st}^{l^*-1}$, P must intersect all but at most $|\mathcal{P}_{ij}^{l^*-1}|/c - 1$ paths in $\mathcal{P}_{ij}^{l^*-1}$ for some ij with $f(ij) \geq l^*$. By the pigeonhole principle, there are at least $|\mathcal{P}_{st}^{l^*-1}|/\binom{h}{2}$ paths (letting these paths $\mathcal{Q}$) in $\mathcal{P}_{st}^{l^*-1}$ each of which intersects all but $|\mathcal{P}_{ij}^{l^*-1}|/c - 1$ paths in $\mathcal{P}_{ij}^{l^*-1}$ for some ij with $f(ij) \geq l^*$ (letting such a set $\mathcal{P}_{ij}^{l^*-1}$ be $\mathcal{P}$).

Then, we have $|\mathcal{Q}| \geq |\mathcal{P}_{st}^{l^*-1}|/\binom{h}{2} \geq \alpha/(c^{2l^*}h^2)$ and $|\mathcal{P}| = \alpha/c^{2l^*+1}$, which implies that $|\mathcal{P}| \geq p'$ and $|\mathcal{Q}| \geq 3k^2|\mathcal{P}|$. Furthermore, by the definitions of $\mathcal{P}$ and $\mathcal{Q}$ and by condition 5, we obtain the following:

1. each path in $\mathcal{Q}$ meets all but at most $|\mathcal{P}|/c \leq |\mathcal{P}|/3k^2$ paths in $\mathcal{P}$, and
2. $\mathcal{P}$ is a Z_1-Z_2 linkage for some $Z_1 \subseteq A_i$ and $Z_2 \subseteq A_j$ such that for each edge $e \in E(\mathcal{P})$, $(\mathcal{P} \cup \mathcal{Q}) - e$ has no Z_1-Z_2 linkage.

This completes the proof of Lemma 5. ◀

Later, we will use this lemma in which $h = k$. The next lemma is a key lemma in this section. Its proof is inspired by [11].

▶ **Lemma 6.** *Suppose that $k, p', \mathcal{P}$, and $\mathcal{Q}$ satisfy the conditions (C1)-(C3) in Lemma 5, and $G = \mathcal{P} \cup \mathcal{Q}$. Each path $P_j \in \mathcal{P}$ has vertices $p_{j,1}, p_{j,2}, \ldots, p_{j,2k}$ which appear in this order from Z_1 to Z_2 such that the following holds:*

(C4) *For all j, let i*th segment *of P_j be the subpath of P_j between $p_{j,i}$ and $p_{j,i+1}$, and let i*th interval *of $\mathcal{P}$ be the union of the ith segment of P_j. Then, for each i, there is a subset $\mathcal{Q}_i \subseteq \mathcal{Q}$ with $|\mathcal{Q}_i| \geq (k-2)|\mathcal{P}|$ such that each path in $\mathcal{Q}_i$ intersects all but at most $|\mathcal{P}|/3k^2$ paths of $\mathcal{P}$ only in their ith segments.*

Proof. Let $p = |\mathcal{P}|$. Since each path in $\mathcal{Q}$ hits all but at most $p/3k^2$ paths, and $|\mathcal{Q}| \geq 3k^2 p$, we may assume that P_1 intersects at least $(1 - 1/3k^2)3k^2 p \geq 2k^2 p$ paths in $\mathcal{Q}$.

Walk along P_1 from one end vertex until encountered kp paths in $\mathcal{Q}$, then pick up $e_1 \in E(P_1) - \bigcup_{Q \in \mathcal{Q}} E(Q)$. Then walk along P_1 until encountered another kp paths in $\mathcal{Q}$, then pick up $e_2 \in E(P_1) - \bigcup_{Q \in \mathcal{Q}} E(Q)$, and so on. Hence we pick up such edges $e_1, e_2, \ldots, e_{2k}$.

By our assumption and Menger's theorem, there exists a vertex set of size at most $p-1$ separating Z_1 and Z_2 in $G - e_i$ for each i. Clearly each path P_j contains exactly one vertex in this cut for $2 \leq j \leq p$. Let $\{p_{2,i}, p_{3,i}, \ldots, p_{2k,i}\}$ be the set of vertices consisting of the cut in $G - e_i$ such that P_j contains $p_{j,i}$ for $2 \leq j \leq p$ and $1 \leq i \leq 2k$. We may define $p_{1,i}$ as one of the end vertices of e_i. Let us define the segment $P_j[i, i+1]$ which is the subpath of P_j between $p_{j,i}$ and $p_{j,i+1}$, for $j = 1, \ldots, p$ and for $i = 1, \ldots, 2k-1$. Note that some of $P_j[i, i+1]$ could be a single vertex. The vertex set $\{p_{1,i}, \ldots, p_{p,i}\}$ divides $\mathcal{P}$ into two parts $\mathcal{P}^{\mathrm{R}_i}$ and $\mathcal{P}^{\mathrm{L}_i}$ such that P^{R_i} is a linkage from Z_1 to $\{p_{1,i}, \ldots, p_{p,i}\}$, and P^{L_i} is a linkage from Z_2 to $\{p_{1,i}, \ldots, p_{p,i}\}$, respectively. Let us remind that at least kp paths in $\mathcal{Q}$ hit $P_1[i, i+1]$ for each i.

Recall that the *ith interval* is defined by $\bigcup_{j=1}^{p} P_j[i, i+1]$. We claim that at least $(k-2)p$ of the kp paths in $\mathcal{Q}$ encountered on $P_1[i, i+1]$ do not leave the ith interval. Since there is no path from Z_1 to Z_2 in $G - \{p_{1,i}, \ldots, p_{p,i}\}$, at most p paths of the kp paths in $\mathcal{Q}$ leave for $\mathcal{P}^{\mathrm{R}_i} - \{p_{1,i}, \ldots, p_{p,i}\}$ through $\{p_{1,i}, \ldots, p_{p,i}\}$. Similarly, at most p paths of the kp paths in $\mathcal{Q}$ leave for $\mathcal{P}^{\mathrm{L}_{i+1}} - \{p_{1,i+1}, \ldots, p_{p,i+1}\}$ through $\{p_{1,i+1}, \ldots, p_{p,i+1}\}$. Therefore, at least $(k-2)p$ of the kp paths in $\mathcal{Q}$ encountered on $P_1[i, i+1]$ do not leave the ith interval. Hence, at least $(k-2)p$ paths in $\mathcal{Q}$ stay strictly inside the ith interval.

Thus, the cuts $\{p_{1,i}, \ldots, p_{p,i}\}$ for $1 \leq i \leq 2k-1$ will break the elements of $\mathcal{P}$ into $2k$ intervals. Moreover, each interval contains at least $(k-2)p$ paths in $\mathcal{Q}$ that stay strictly in the interval. These paths form the set $\mathcal{Q}_i$. This completes the proof. ◀

4 Main Lemmas

Suppose that $\mathcal{P}$ and $\mathcal{Q}$ are linkages satisfying the conditions (C1)-(C4) in Lemmas 5 and 6, and let $G = \mathcal{P} \cup \mathcal{Q}$ and $p = |\mathcal{P}|$. For each $i = 1, \ldots, 2k$, define G'_i to be the induced subgraph of G in the ith interval. We say that an index set $X \subseteq \{1, 2, \ldots, p\}$ is *good in G'_i* if it satisfies the following: for any subsets $Y_1, Y_2 \subseteq X$ with $|Y_1| = |Y_2| = 2r$, there are $2r$ mutually vertex-disjoint paths from $\{p_{j,i} \mid j \in Y_1\}$ to $\{p_{j,i+1} \mid j \in Y_2\}$ in G'_i.

Our first lemma in this section is the following.

▶ **Lemma 7.** *Let r and k be integers, and set $p' = 400k^2 r$. Suppose that $\mathcal{P}$ and $\mathcal{Q}$ are linkages that satisfy conditions (C1)-(C4) in Lemmas 5 and 6, and let $p = |\mathcal{P}|$. For each i, there is a good set X_i in G'_i with $|X_i| \geq 3p/4$. Moreover, $|X_{i-1} \cap X_i \cap X_{i+1}| \geq 100k^2 r$ for $i = 2, \ldots, 2k-1$.*

Proof. Define $X = \{j \mid P_j \in \mathcal{P}$ hits at least $2r$ paths of $\mathcal{Q}_i\}$. Then, by simple counting argument, we have $|X| \geq 3p/4$, because $(p - p/3k^2)(k-2)p > (k-2)p(3p/4) + 2r(p/4)$.

Assume that X is not a good set in G'_i. Then, for some subsets $Y_1, Y_2 \subseteq X$ with $|Y_1| = |Y_2| = 2r$, there is a separation (A, B) of order at most $2r-1$ in G'_i with $\{p_{j,i} \mid j \in Y_1\} \subseteq V(A)$ and $\{p_{j,i+1} \mid j \in Y_2\} \subseteq V(B)$. We now consider $Z_A := \{j \mid V(P_j) \cap V(A - B) \neq \emptyset\}$ and $Z_B := \{j \mid V(P_j) \cap V(B - A) \neq \emptyset\}$. Since $P_j \in \mathcal{P}$ hits at least $2r > |V(A) \cap V(B)|$ paths of $\mathcal{Q}_i$ for each $j \in X$ and moreover each path in $\mathcal{Q}$ intersects at least $(1 - 1/3k^2)p \geq 3p/4$ paths of $\mathcal{P}$, both $|Z_A|$ and $|Z_B|$ are at least $3p/4$. Since $|Z_A \cap Z_B| \leq |V(A) \cap V(B)| \leq 2r - 1$, we have $|Z_A \cup Z_B| = |Z_A| + |Z_B| - |Z_A \cap Z_B| > p$, which is a contradiction.

Since $|X_i| \geq 3p/4$ for each i, we have $|X_{i-1} \cap X_i \cap X_{i+1}| \geq p - 3 \cdot (p/4) \geq 100k^2 r$. ◀

We say that a *leaf* of a connected graph is a vertex of degree one, and a $K_{1,k}$-minor (or a *k-star-minor*) is a connected subgraph with at least k leaves. For a linkage $\mathcal{P}' = \{P'_1, \ldots, P'_{|\mathcal{P}'|}\}$ in a graph G, a $K_{1,k}$-minor S in G is said to be *attached to* $\mathcal{P}'$ if every leaf of S is contained in $\mathcal{P}'$, and $|V(S) \cap V(P'_j)| = 1$ holds whenever $V(P'_j)$ contains a leaf of S.

The next lemma is the key lemma in our proof. It roughly says that one can either find an $r \times r$-grid minor in G'_i or else, given a good set X in G'_i, construct a minor of a "star-like graph" with at least k leaves in X. This gives us a $K_{1,k}$-minor with some condition in G'_i. This lemma allows us to "weave" the paths in $\mathcal{P}$ and construct a K_k-minor. Some idea in our proof can be found in [2].

▶ **Lemma 8.** *For each i, we have the following. Let $r, k, p, \mathcal{P}$, $\mathcal{Q}$, and G'_i be as above, and let X be a good set in G'_i with $|X| \geq 100k^2 r$. Then, either*

1. *G'_i has an $r \times r$-grid minor, or*
2. *there exist $Y_1, Y_2 \subseteq X$ with $|Y_1| = |Y_2| = k$ such that G'_i has a linkage $\mathcal{P}'$ from $\{p_{j,i} \mid j \in Y_1\}$ to $\{p_{j,i+1} \mid j \in Y_2\}$ and a $K_{1,k}$-minor S' attached to $\mathcal{P}'$.*

Proof. Since we only consider the ith interval of $\mathcal{P}$, we omit the index i in this proof for simplicity if no confusion may arise. That is, we denote $P_j[i, i+1]$ and G'_i by P_j and G', respectively.

Let $\mathcal{P}_X = \bigcup_{j \in X} P_j$ be the linkage that consists of the paths corresponding to X. Since X is a good set in G', we shall only focus on the unique connected component of G' containing $\mathcal{P}_X$. For our convenience, let us assume that G' itself is such a unique component.

Let Y be the set of connected components of $G' - \mathcal{P}_X$. We consider the auxiliary graph G^* with the vertex set $X \cup Y$ such that there exists an edge connecting $j \in X$ and $y \in Y$ if a $\mathcal{P}_X$-bridge y has attachments in P_j, and there exists an edge connecting $j_1, j_2 \in X$ if G' has an edge connecting P_{j_1} and P_{j_2}. We note that G^* is connected, since we assume the connectivity of G'.

We say that a $K_{1,t}$-minor S' in G^* with t leaves is a *good $K_{1,t}$-minor* if all leaves are in X and $|V(S') \cap X| \leq 3t$. We take disjoint subgraphs $S_1, \ldots, S_l$ in G^* such that S_i is a good K_{1,t_i}-minor with $t_i \geq 3$ for $i = 1, \ldots, l$, and

the total number of leaves $\sum_{i=1}^{l} t_i$ is as large as possible.

We show the following claim.

▶ **Claim 9.** *If $\sum_{i=1}^{l} t_i \geq 3k$, then there is a $K_{1,k}$-minor S^* in G^* such that all the leaves of S^* are in X.*

Proof. For any two subgraphs S_i, S_j with t_i, t_j leaves, respectively, if there is a path between S_i and S_j, then we can obtain a K_{1,t_i+t_j-2}-minor whose all leaves are in X. Note that $t_i + t_j - 2 > t_i$ and $t_i + t_j - 2 > t_j$.

Having proved this, we just greedily construct a star-minor such that all the leaves of the star-minor are in X. At the first step, we pick up one graph $S_i \in \{S_1, \ldots, S_l\}$. Then

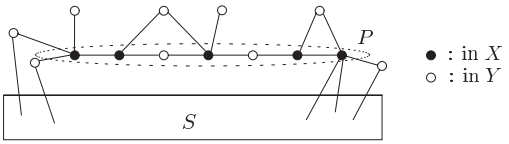


Figure 1 A connected component of $G^* - S$

we find a path between S_i and $\{S_1, \ldots, S_l\} \setminus \{S_i\}$. Such a path must exist because G^* is connected. Suppose that the path connects S_i and S_j with $i \neq j$. Then we merge S_i and S_j as above to obtain a K_{1,t_i+t_j-2}-minor with all the leaves in X. Next, we find a path between the K_{1,t_i+t_j-2}-minor and $\{S_1, \ldots, S_l\} \setminus \{S_i, S_j\}$, and we repeat this process until the end.

By the above remark, in each iteration, we get a star-minor with more leaves (in X) than the star-minor in the previous iteration. In fact, since the total number of leaves $\sum_{i=1}^{l} t_i$ is at least $3k$ at the beginning, in the final iteration, we get a star-minor with at least $\sum_{i=1}^{l}(t_i - 2) \geq k$ leaves in X. Note that we use the assumption $t_i \geq 3$ in this inequality. This completes the proof of Claim 9. ◀

We note that if there is a $K_{1,k}$-minor S^* in G^* such that all the leaves are in X, then we have the second conclusion of Lemma 8, in which $\mathcal{P}' = \mathcal{P}_X$ and S' is a minimal subgraph corresponding to S^*. Hence, in what follows, we assume that $\sum_{i=1}^{l} t_i < 3k$. By the definition of a good $K_{1,t}$-minor, this implies that $|V(S) \cap X| < 9k$ for $S := \bigcup_{i=1}^{l} S_i$. Now we show the following.

▶ **Claim 10.** *Let $S = \bigcup_{i=1}^{l} S_i$. As shown in Figure 1, each connected component of $G^* - S$ consists of a path P, a vertex set $Y' \subseteq Y - V(S)$, and edges between $V(P)$ and Y' such that for every $y \in Y'$, either*

- *$N_{G^*}(y) \cap V(P)$ consists of one vertex, or*
- *$N_{G^*}(y) \cap V(P)$ consists of two vertices v_1, v_2 with either $v_1v_2 \in E(P)$ or $v_1v_3, v_3v_2 \in E(P)$ for some $v_3 \in V(P) \cap Y$.*

Furthermore, each internal vertex of P is not adjacent to S, and each vertex in Y' adjacent to an internal vertex of P is not adjacent to S.

Proof. Let C be a connected component of $G^* - S$. If $|V(C) \cap X| \leq 2$, then the claim is obvious, because each vertex $y \in Y$ is not adjacent to a vertex in Y by the definition of G^*.

Suppose that $|V(C) \cap X| \geq 3$. By our choice of $\{S_1, \ldots, S_l\}$, we observe that

- each vertex $y \in V(C) \cap Y$ is adjacent to at most two vertices in $V(C)$, and
- if a vertex $y \in V(C) \cap Y$ is adjacent to two vertices in $V(C)$, then y is not adjacent to S.

Again, we note that each vertex $y \in Y$ is not adjacent to a vertex in Y. While C contains a vertex $y \in V(C) \cap Y$ that is adjacent to two vertices v_1, v_2 in $V(C)$, we remove y (together with edges yv_1 and yv_2) and add an edge v_1v_2. Then, the obtained graph C' contains vertices in Y of degree one and vertices in X.

If there exists a vertex $x \in V(C') \cap X$ adjacent to three vertices in $V(C') \cap X$, then by adding this $K_{1,3}$-minor to S, we obtain a new set of star-minors with more total number of leaves, which contradicts the choice of S. Hence, the subgraph of C' induced by $V(C') \cap X$ forms a path or a cycle with multiple edges. Let $x_1, x_2, \ldots, x_q$ be vertices of $V(C') \cap X$ that appear along this path (or cycle) in this order.

If x_j is adjacent to a vertex v in S for some $j = 2, 3, \ldots, q-1$, then we can increase the total number of leaves of S by adding a $K_{1,3}$-minor whose leaves are x_{j-1}, x_{j+1}, and v. Therefore, x_j is not adjacent to S for $j = 2, 3, \ldots, q-1$. Similarly, if there exists a vertex

$y \in V(C') \cap Y$ that is adjacent to x_j for some $j = 2, 3, \ldots, q-1$, then y is not adjacent to S. Note that, by this argument, we can also see that the subgraph of C' induced by $V(C') \cap X$ is not a cycle but a path, because G^* is connected.

Since each vertex $y \in Y$ is not adjacent to a vertex in Y, the original component C is obtained from C' by subdividing some edges into two edges. Thus, the claim holds by the above properties of C'. ◀

▶ **Claim 11.** *Suppose that $|V(S) \cap X| < 9k$. Then, some connected component of $G^* - S$ contains at least $4r + 2k(r+4)$ vertices in X.*

Proof. Let $\mathcal{C}$ be the set of connected components of $G^* - S$ each containing a vertex in X. By the choice of S, we can see that following:

- for each $x \in V(S) \cap X$, x is adjacent to at most one component of $\mathcal{C}$, and
- for each $y \in V(S) \cap Y$, y is adjacent to no component of $\mathcal{C}$.

This means that $|\mathcal{C}| \leq |V(S) \cap X| < 9k$, because G^* is connected. Since $|X| \geq 100k^2 r > 9k \cdot (4r + 2k(r+4))$, at least one connected component of $G^* - S$ contains at least $4r + 2k(r+4)$ vertices in X. ◀

By Claims 10 and 11, we can see that $G^* - S$ contains a long path. The following claim shows that each subgraph of G' corresponding to a long path with some condition contains either an $r \times r$-grid minor or "crossing paths".

▶ **Claim 12.** *Suppose that $0, 1, 2, \ldots, r+3 \in X$ appear in a path of $G^* - S$ in this order, and suppose also that there exist mutually vertex-disjoint paths $R_1, \ldots, R_r$ from $V(P_1)$ to $V(P_{r+2})$ in $G' - (P_0 \cup P_{r+3})$. Then, either G' contains an $r \times r$-grid minor or there exist two vertex-disjoint paths P' and R' in $G' - (P_0 \cup P_{r+3})$ such that P' connects $p_{j_1,i}$ and $p_{j_2,i+1}$ for some $j_1, j_2 \in \{2, 3, \ldots, r+1\}$, P' does not intersect with $V(P_1) \cup V(P_{r+2})$, and R' connects $V(P_1)$ and $V(P_{r+2})$. Furthermore, if such paths P' and R' exist, then $G' - (P_0 \cup P_{r+3})$ contains a linkage $\mathcal{P}' = \{P_1, P_{r+2}, P'\}$ and a $K_{1,3}$-minor attached to $\mathcal{P}'$.*

Proof. By the latter half of Claim 10, each of $R_1, \ldots, R_r$ intersects with $P_1, P_2, \ldots, P_{r+2}$ but does not intersect with the subgraph corresponding to S. Let D be the graph obtained from $(\bigcup_{1 \leq j \leq r+2} P_j) \cup (\bigcup_{1 \leq i \leq r} R_i)$ by executing the following procedure: contract P_1 to a single vertex s_1, contract P_{r+2} to a single vertex t_1, add a vertex s_2 and edges $s_2 p_{j,i}$ for $j = 2, 3, \ldots, r+1$, and add a vertex t_2 and edges $t_2 p_{j,i+1}$ for $j = 2, 3, \ldots, r+1$ (see Figure 2). Then, by a characterization of the existence of 2 vertex-disjoint paths (see [30]), either there exist a s_1-t_1 path and a s_2-t_2 path that are mutually vertex-disjoint, or D contains pairwise disjoint vertex sets $U_1, \ldots, U_q$ $(q \geq 0)$ containing none of $\{s_1, t_1, s_2, t_2\}$ such that

(1) for $1 \leq i, j \leq q$ with $i \neq j$, $N_D(U_i) \cap U_j = \emptyset$,
(2) for $1 \leq i \leq q$, $|N_D(U_i)| \leq 3$, and
(3) if $\bar{D}$ is the graph obtained from D by contracting each component U_i to a single vertex for each i, then $\bar{D}$ can be embedded in a plane so that s_1, s_2, t_1 and t_2 are on the outer face boundary in this order.

If there exist a s_1-t_1 path and a s_2-t_2 path that are mutually vertex-disjoint, then the corresponding paths are two vertex-disjoint paths P' and R' in $G' - (P_0 \cup P_{r+3})$ such that P' connects $p_{j_1,i}$ and $p_{j_2,i+1}$ for some $j_1, j_2 \in \{2, 3, \ldots, r+1\}$, P' does not intersect with $V(P_1) \cup V(P_{r+2})$, and R' connects $V(P_1)$ and $V(P_{r+2})$. The existence of a $K_{1,3}$-minor attached to $\mathcal{P}' = \{P_1, P_{r+2}, P'\}$ is guaranteed by the existence of R'.

Suppose that there exist disjoint vertex sets $U_1, \ldots, U_q$ $(q \geq 0)$ as above. By the construction of $\bar{D}$ in the condition (3), the paths in $\bar{D}$ corresponding to $P_2, \ldots, P_{r+1}$ are

mutually vertex-disjoint except their end points, and the same thing holds for the paths in $\bar{D}$ corresponding to $R_1, \ldots, R_r$. By the planarity of $\bar{D}$, these paths form an $r \times r$-grid minor (see [23]). Since G' contains $\bar{D}$ as a minor, we have an $r \times r$-grid minor of G'. ◀

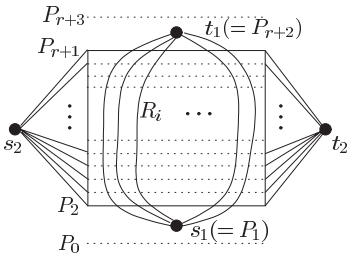


Figure 2 Construction of D

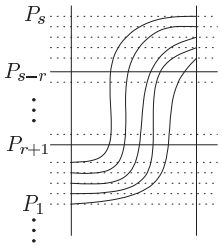


Figure 3 Paths from P_{r+1} to P_{s-r}

Now we are ready to prove Lemma 8. By Claims 10 and 11, $G^* - S$ contains a path containing at least $4r + 2k(r+4)$ vertices in X. We may assume that $-s, \ldots, -2, -1, 1, 2, \ldots, s \in X$ appear in the path in this order, where $s = 2r + k(r+4)$. Since X is a good set, there are $2r$ mutually vertex-disjoint paths from $\{p_{j,i} \mid j \in \pm\{1, 2, \ldots, r\}\}$ to $\{p_{j,i+1} \mid j \in \pm\{s, s-1, \ldots, s-r+1\}\}$. By the latter half of Claim 10, this means that G' contains either r vertex-disjoint paths from P_{r+1} to P_{s-r} or r vertex-disjoint paths from P_{-r-1} to P_{-s+r} that do not intersect with the subgraph corresponding to S. By symmetry, we may assume that G' contains r vertex-disjoint paths from P_{r+1} to P_{s-r} that do not intersect with the subgraph corresponding to S (see Figure 3).

We partition $\{r+1, r+2, \ldots, s-r\}$ into k disjoint sets $U_1, U_2, \ldots, U_k$ by setting $U_l := \{l(r+4) - 3, l(r+4) - 2, \ldots, l(r+4) + r\}$. Note that $|U_l| = r + 4$ for each l. Then, by the assumption that $1, 2, \ldots, s \in X$ appear in this order, there exist r vertex-disjoint paths from $P_{l(r+4)-2}$ to $P_{l(r+4)+r-1}$ in $G' - (P_{l(r+4)-3} \cup P_{l(r+4)+r})$ for each l. We now apply Claim 12 for each U_l. If we can find an $r \times r$-grid minor for some l, then we are done. Otherwise, by Claim 12, for each l, we can take a linkage $\mathcal{P}'_l = \{P_{l(r+4)-2}, P_{l(r+4)+r-1}, P'_l\}$ and a $K_{1,3}$-minor attached to $\mathcal{P}'_l$.

Let $\mathcal{P}' = \bigcup_l \mathcal{P}'_l$. Then, we have k disjoint $K_{1,3}$-minors attached to $\mathcal{P}'$. Since the total number of leaves is $3k$, by the same argument as Claim 9, we can construct a $K_{1,k}$-minor attached to $\mathcal{P}'$ that is the second conclusion of Lemma 8. ◀

5 Main Proof

In this section, we give a proof of Theorem 1. That is, we show that there exists a constant $w = |V(H)|^{O(|E(H)|)} \cdot r$ such that every graph with treewidth at least w has either an H-minor or an $r \times r$-grid minor.

By applying Lemma 5 with $k = h = |V(H)|$ and $p' = 400k^2 r$, we obtain an integer $w = k^{O(|E(H)|)} \cdot r$. If the graph G has treewidth at least w, then either G contains an H-minor or two linkages $\mathcal{P}$ and $\mathcal{Q}$ satisfying (C1)-(C3). By Lemma 6, the linkage $\mathcal{P}$ can be partitioned into $2k$ intervals with the condition (C4). By Lemma 7, for each $i = 1, 2, \ldots, 2k$, there is a good set X_i in G'_i such that $|X_i| \geq 3p/4$ and $|X_{i-1} \cap X_i \cap X_{i+1}| \geq 100k^2 r$.

For each $i = 1, 3, 5, \ldots, 2k-1$, we apply Lemma 8 with $X = X_{i-1} \cap X_i \cap X_{i+1}$ (where we define $X_0 = \{1, 2 \ldots, p\}$). If an $r \times r$-grid minor is obtained, then we are done. Thus, we may assume that there exist $Y_{1,i}, Y_{2,i} \subseteq X_{i-1} \cap X_i \cap X_{i+1}$ with $|Y_{1,i}| = |Y_{2,i}| = k$ such that G'_i has a linkage $\mathcal{P}'_i$ from $\{p_{j,i} \mid j \in Y_{1,i}\}$ to $\{p_{j,i+1} \mid j \in Y_{2,i}\}$ and a $K_{1,k}$-minor S'_i attached to $\mathcal{P}'_i$.

For $i = 2, 4, 6, \ldots, 2k-2$, since $Y_{2,i-1}, Y_{1,i+1} \subseteq X_i$, there exist k vertex-disjoint paths from $\{p_{j,i} \mid j \in Y_{2,i-1}\}$ to $\{p_{j,i+1} \mid j \in Y_{1,i+1}\}$ by the definition of good sets. That is, we can connect $\mathcal{P}'_{i-1}$ and $\mathcal{P}'_{i+1}$ in the ith interval. By adding these $(k-1) \times k$ paths to $\bigcup_i \mathcal{P}'_i$, we obtain a linkage $\mathcal{P}'$ from $Y_{1,1}$ to $Y_{2,2k-1}$ and $K_{1,k}$-minors $S'_1, S'_3, \ldots, S'_{2k-1}$ attached to $\mathcal{P}'$. This graph contains a complete bipartite graph $K_{k,k}$ as a minor, which implies that it contains a K_k-minor. Since H is a subgraph of K_k, this completes the proof of the first half of Theorem 1.

By following the above arguments, Lemmas 6, 7, and 8 can be translated in polynomial time algorithms by using known algorithms for finding constant number of disjoint paths between two disjoint sets, for finding a minimum vertex cut, and for solving the 2 paths problem (e.g. [30, 31, 32]). We note that, in the proof of Lemma 8, we do not have to maximize the total number of leaves $\sum_{i=1}^{l} t_i$ at the beginning. This is because, if we cannot obtain the desired objects in Claims 10, 11, and 12, then we can find a set of star-minors with more total number of leaves. Hence, we only have to apply these claims, repeatedly.

To translate Lemma 5 to a polynomial time algorithm, it suffices to translate Lemmas 3 and 4 to polynomial time algorithms. Given a tree T and a vertex set $X \subseteq V(T)$, we can easily find an edge set F as in Lemma 4 in linear time by a simple greedy algorithm. On the other hand, we have no polynomial time algorithm to compute either a tree decomposition of G of width $< \alpha + \beta - 1$ or an α-mesh of order β in G as in Lemma 3. However, by the arguments in [11] (see also [21, Lemma 3.10]), we can find in polynomial time either a tree decomposition of G of width $< w$ or h vertex sets $A_1, \ldots, A_h$ as in the proof of Lemma 5. Therefore, all the procedures in the proof can be done in polynomial time in n and w. ◀

References

1 S. Arnborg and A. Proskurowski, Linear time algorithms for NP-hard problems restricted to partial k-trees, *Discrete Appl. Math.*, **23** (1989), 11–24.

2 T. Böhme, K. Kawarabayashi, J. Maharry and B. Mohar, Linear connectivity forces large complete bipartite graph minors, *J. Combin. Theory Ser. B*, **99** (2009), 557–582.

3 C. Chekuri, S. Khanna and B. Shepherd, Edge-disjoint paths in planar graphs with constant congestion, *SIAM J. Comput.*, **39** (2009), 281–301.

4 B. Courcelle, Graph rewriting: An algebraic and logic approach, in *Handbook of Theoretical Computer Science* **2**, Elsevier, 1990, 194–242.

5 E.D. Demaine, F.V. Fomin, M. Hajiaghayi, and D.M. Thilikos, Bidimensional parameters and local treewidth, *SIAM J. Discrete Mathematics*, **18** (2004), 501–511.

6 E.D. Demaine, F.V. Fomin, M. Hajiaghayi and D.M. Thilikos, Subexponential parameterized algorithms on graphs of bounded genus and H-minor-free graphs, *J. ACM*, **52** (2005), 866–893.

7 E.D. Demaine and M. Hajiaghayi. Bidimensionality: New connections between FPT algorithms and PTASs, *Proc. 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2005, 590–601.

8 E.D. Demaine and M. Hajiaghayi, Linearity of grid minors in treewidth with applications through bidimensionality, *Combinatorica*, **28** (2008), 19–36.

9 E.D. Demaine, M. Hajiaghayi and K. Kawarabayashi, Algorithmic graph minor theory: Decomposition, approximation, and coloring, *Proc. the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, 2005, 637–646

10 E.D. Demaine, M. Hajiaghayi and K. Kawarabayashi, Algorithmic graph minor theory: Improved grid minor bounds and Wagner's contraction, *Algorithmica*, **54** (2009), 142–180.

11 R. Diestel, K. Yu. Gorbunov, T.R. Jensen and C. Thomassen, Highly connected sets and the excluded grid theorem, *J. Combin. Theory Ser. B*, **75** (1999), 61–73.

12 M. Grohe, The complexity of homomorphism and constraint satisfaction problems seen from the other side, *J. ACM*, **54** (2007), 1–24.

13 N. Garg, V. Vazirani and M. Yannakakis, Primal-dual approximation algorithms for integral flow and multicut in trees with applications to matching and set cover, *Algorithmica*, **18** (1997), 3–20.

14 K. Kawarabayashi and B. Reed, A nearly linear time algorithm for the half disjoint paths packing, *Proc. the 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2008, 446–454.

15 K. Kawarabayashi and Y. Kobayashi, The edge disjoint paths problem in Eulerian graphs and 4-edge-connected graphs, *Proc. the 21st Annual ACM-SIAM Symposium on Discrete Algorithms, (SODA)*, 2010, 345–353.

16 K. Kawarabayashi, Y. Kobayashi and B. Reed, The disjoint paths problem in quadratic time, *J. Combin. Theory Ser. B*, to appear.

17 K. Kawarabayashi and P. Wollan, A shorter proof of the graph minor algorithm - The unique linkage theorem -, *Proc. the 42nd ACM Symposium on Theory of Computing (STOC)*, 2010, 687–694.

18 K. Kawarabayashi and P. Wollan, A simpler algorithm and shorter proof for the graph minor decomposition, *Proc. the 43rd ACM Symposium on Theory of Computing (STOC)*, 2011, 451–458.

19 K. Kawarabayashi, S. Norine, R. Thomas and P. Wollan, K_6 minors in 6-connected graphs of bounded tree width, *submitted*.

20 J. Kleinberg, Decision algorithms for unsplittable flows and the half-disjoint paths problem, *Proc. the 30th ACM Symposium on Theory of Computing (STOC)*, 1998, 530–539.

21 S. Kreutzer and S. Tazari, On brambles, grid-like minors and parameterized intractability of monadic second-order logic, *Proc. the 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2010, 354–362.

22 B. Reed, Tree width and tangles: a new connectivity measure and some applications, in *Surveys in Combinatorics*, London Math. Soc. Lecture Note Ser. **241**, Cambridge Univ. Press, Cambridge, 1997, 87–162.

23 N. Robertson and P.D. Seymour, Graph minors. III. Planar tree-width, *J. Combin. Theory Ser. B*, **36** (1984), 49–64.

24 N. Robertson and P.D. Seymour, Graph minors. V. Excluding a planar graph, *J. Combin. Theory Ser. B*, **41** (1986), 92–114.

25 N. Robertson and P.D. Seymour, Graph minors. XIII. The disjoint paths problem, *J. Combin. Theory Ser. B*, **63** (1995), 65–110.

26 N. Robertson and P.D. Seymour, Graph minors. XX. Wagner's conjecture, *J. Combin. Theory Ser. B*, **92** (2004), 325–357.

27 N. Robertson and P.D. Seymour, Graph Minors. XXI. Graphs with unique linkages, *J. Combin. Theory Ser. B*, **99** (2009), 583–616.

28 N. Robertson and P.D. Seymour, Graph minors. XXII. Irrelevant vertices in linkage problems, to appear in *J. Combin. Theory Ser. B*.

29 N. Robertson, P.D. Seymour and R. Thomas, Quickly excluding a planar graph, *J. Combin. Theory Ser. B*, **62** (1994), 323–348.

30 P.D. Seymour, Disjoint paths in graphs, *Discrete Math.*, **29** (1980), 293–309.

31 T. Tholey, Solving the 2-disjoint paths problem in nearly linear time, *Theory of computing systems*, **39** (2006), 51–78.

32 C. Thomassen, 2-linked graph, *European Journal of Combinatorics*, **1** (1980), 371–378.

Linear-Space Data Structures for Range Mode Query in Arrays*

Timothy M. Chan[1], Stephane Durocher[2], Kasper Green Larsen[3], Jason Morrison[2], and Bryan T. Wilkinson[1]

1 **University of Waterloo, Waterloo, Canada,** tmchan@uwaterloo.ca **and** b3wilkin@uwaterloo.ca
2 **University of Manitoba, Winnipeg, Canada,** durocher@cs.umanitoba.ca **and** jason_morrison@umanitoba.ca
3 **Aarhus University, Aarhus, Denmark,** larsen@cs.au.dk

Abstract

A mode of a multiset S is an element $a \in S$ of maximum multiplicity; that is, a occurs at least as frequently as any other element in S. Given an array $A[1:n]$ of n elements, we consider a basic problem: constructing a static data structure that efficiently answers range mode queries on A. Each query consists of an input pair of indices (i, j) for which a mode of $A[i:j]$ must be returned. The best previous data structure with linear space, by Krizanc, Morin, and Smid (ISAAC 2003), requires $O(\sqrt{n} \log \log n)$ query time. We improve their result and present an $O(n)$-space data structure that supports range mode queries in $O(\sqrt{n/\log n})$ worst-case time. Furthermore, we present strong evidence that a query time significantly below $\sqrt{n}$ cannot be achieved by purely *combinatorial* techniques; we show that boolean matrix multiplication of two $\sqrt{n} \times \sqrt{n}$ matrices reduces to n range mode queries in an array of size $O(n)$. Additionally, we give linear-space data structures for orthogonal range mode in higher dimensions (queries in near $O(n^{1-1/2d})$ time) and for halfspace range mode in higher dimensions (queries in $O(n^{1-1/d^2})$ time).

1998 ACM Subject Classification E.1 Data Structures

Keywords and phrases mode, range query, data structure, linear space, array

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.290

1 Introduction

The *frequency* of an element x in a multiset S, denoted $\text{freq}_S(x)$, is the number of occurrences (i.e., the multiplicity) of x in S. A *mode* of S is an element $a \in S$ such that for all $x \in S$, $\text{freq}_S(x) \leq \text{freq}_S(a)$. A multiset S may have multiple distinct modes; the frequency of the modes of S, denoted by m, is unique.

Along with the mean and median, the mode is a fundamental statistic in data analysis. Given a sequence of n elements ordered in a list A, a range query seeks to compute the corresponding statistic on the multiset determined by a subinterval of the list: $A[i:j]$. The objective is to preprocess A to construct a data structure that supports efficient response to one or more subsequent range queries, where the corresponding input parameters (i, j) are provided at query time. Such a data structure is useful as it allows us to report statistics over any window of a given sequence of data.

* Work supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), and in part by MADALGO – Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 290–301

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

We assume the standard RAM model of computation with word size $w = \Omega(\log n)$. Although the complete set of possible queries can be precomputed and stored using $\Theta(n^2)$ space, practical data structures require less storage while still enabling efficient response time. For all i, if $i = j$, then a range query must report $A[i]$. Consequently, any range query data structure for a list of n items requires $\Omega(n)$ storage space in the worst case [2]. This leads to a natural question: how quickly can an $O(n)$-space data structure answer range queries? A range mean query is equivalent to a normalized range sum query (partial sum query), for which a precomputed prefix-sum array provides a linear-space static data structure with constant query time [16]. Range median queries have been analyzed extensively in recent years and are closely related to range counting, where efficient data structures are now known (with linear space and logarithmic or slightly sublogarithmic query time) [2, 3, 5, 10, 11, 14, 16, 20, 21]. In contrast, range mode queries appear more challenging than range mean and median. As expressed recently by Brodal et al. [3, page 2]: "The problem of finding the most frequent element within a given array range is still rather open."

The best previous linear-space data structure for range mode query was by Krizanc et al. [15, 16], who obtained a query time of $O(\sqrt{n}\log\log n)$.[1] No better approaches have been discovered in the intervening eight years, which leads one to suspect that a $\sqrt{n}$-type bound might be the best one could hope for.

Indeed, we present strong evidence that purely combinatorial approaches cannot avoid the $\sqrt{n}$ effect in the preprocessing or query costs, up to polylogarithmic factors. (Krizanc et al.'s method has near $n^{3/2}$ preprocessing time.) More specifically, we show in Section 7 that boolean matrix multiplication (matrix multiplication on $\{0,1\}$-matrices with addition and multiplication replaced by OR and AND, respectively) of two $\sqrt{n}\times\sqrt{n}$ matrices reduces to n range mode queries in an array of size $O(n)$. This reduction implies that any data structure for range mode must have either $\Omega(n^{\omega/2})$ preprocessing time or $\Omega(n^{\omega/2-1})$ query time in the worst case, where ω denotes the matrix multiplication exponent. Since the current best matrix multiplication algorithm has exponent 2.3727 [23], we cannot obtain preprocessing time better than $n^{1.18635}$ and query time better than $n^{0.18635}$ simultaneously with current knowledge. Moreover, since the current best *combinatorial* algorithm for boolean matrix multiplication (which avoids algebraic techniques as in Strassen's) has running time only a polylogarithmic factor better than cubic [1], we cannot obtain preprocessing time better than $n^{3/2}$ and query time better than $\sqrt{n}$ simultaneously by purely combinatorial techniques with current knowledge, except for polylogarithmic-factor speedups.

In view of the above hardness result, it is therefore worthwhile to pursue more modest improvements for the range mode problem. Notably, can the extra $\log\log$ factor in Krizanc et al.'s bound be eliminated?

In Section 3, we give a data structure that accomplishes just that: with $O(n)$ space, we can answer range mode queries in $O(\sqrt{n})$ time. The data structure is based on—and in some ways simplifies—Krizanc et al.'s, since we use only rudimentary structures (mostly arrays), without van Emde Boas trees or repeated binary searches.

In fact, we go beyond eliminating a mere $\log\log$ factor: in Section 6, we present an $O(n)$-space data structure that answers range mode queries in $o(\sqrt{n})$ time. The precise worst-case time bound is $O(\sqrt{n/w}) \subseteq O(\sqrt{n/\log n})$. As one might guess, bit packing tricks are used to achieve the speedup, but in addition we need a nontrivial combination of ideas, including

[1] The original data structure described by Krizanc et al. [16] supports queries in $O(\sqrt{n}\log n)$ time. As they remarked, this time can be reduced to $O(\sqrt{n}\log\log n)$ by using van Emde Boas trees for predecessor search [22].

partitioning elements into two sets (one with small maximum frequency and another with a small number of distinct elements), each handled by a different method, and an interesting application of rank/select data structures (from the world of succinct data structures).

In Section 8, we consider a natural higher-dimensional generalization of the problem: given a set of coloured points in $\mathbb{R}^d$, support queries for the most frequently occurring colour in some query range. We obtain the first nontrivial results for this geometric problem. For example, for orthogonal ranges, we give a near-linear space data structure that supports queries in near $O(n^{1-1/2d})$ time. For halfspace ranges, we give a linear-space data structure that supports queries in $O(n^{1-1/d^2})$ time. This latter result is obtained using an interesting application of geometric *cuttings* [7], in addition to standard range searching data structures.

Throughout the paper, let m denote the maximum frequency (i.e., the mode of the overall array), and let Δ denote the number of distinct elements ($m, \Delta \leq n$).

2 Related Work

Computing a Mode. The mode of a multiset S of n items can be found in $O(n \log n)$ time by sorting S and scanning the sorted list to identify the longest sequence of identical elements. By reduction from element uniqueness, a matching $\Omega(n \log n)$ lower bound in the comparison model follows. Better bounds on the worst-case time can be obtained by parameterizing in terms of m or Δ. A worst-case time of $O(n \log \Delta)$ is easily achieved by inserting the n elements into a balanced search tree in which each node stores a key and its frequency. Munro and Spira [19] described an $O(n \log(n/m))$-time algorithm and a matching comparison-based lower bound. On the word RAM model, the mode can be computed in linear expected time by hashing.

Range Mode Query. As mentioned, a data structure of Krizanc et al. [16] requires linear space and provides $O(\sqrt{n} \log \log n)$ query time. Krizanc et al. also considered larger-space structures. They described data structures that provide constant-time queries using $O(n^2 \log \log n / \log n)$ space and $O(n^\epsilon \log n)$-time queries using $O(n^{2-2\epsilon})$ space, for any fixed $\epsilon \in (0, 1/2]$. Petersen and Grabowski [21] improved the first bound to constant time and $O(n^2 \log \log n / \log^2 n)$ space and Petersen [20] improved the second bound to $O(n^\epsilon)$-time queries using $O(n^{2-2\epsilon})$ space, for any fixed $\epsilon \in [0, 1/2)$. Although its space requirement is almost linear in n as ϵ approaches $1/2$, the data structure of Petersen [20] requires $\omega(n)$ space (the number of levels in a hierarchical set of tables and hash functions approaches ∞ as $\epsilon \to 1/2$). Our new approach can also lead to improved space-time tradeoffs (see the statement of Theorem 7 with the parameter $s = n^{1-\epsilon}$): we can obtain $O(n^\epsilon)$ query time with $O(n^{2-2\epsilon} / \log n)$ space for any fixed $\epsilon \in [0, 1/2]$. This improves Petersen's result (though for $\epsilon = 0$, Petersen and Grabowski's result remains slightly better). Finally, Greve et al. [12] prove a lower bound of $\Omega(\log n / \log(s \cdot w/n))$ query time for any data structure that uses s memory cells of w bits in the cell probe model.

Other Query Problems. Bose et al. [2] considered approximate range mode queries, in which the objective is to return an element whose frequency is at least αm. They gave a data structure that requires $O(n/(1-\alpha))$ space and answers approximate range mode queries in $O(\log \log_{1/\alpha} n)$ time for any fixed $\alpha \in (0, 1)$, as well as data structures that provide constant-time queries for $\alpha \in \{1/2, 1/3, 1/4\}$, using space $O(n \log n)$, $O(n \log \log n)$, and $O(n)$, respectively. Greve et al. [12] gave data structures that support approximate range mode queries in $O(1)$ time using $O(n)$ space for $\alpha = 1/3$, and $O(\log(\alpha/(1-\alpha)))$ time using $O(n\alpha/(1-\alpha))$ space for any fixed $\alpha \in [1/2, 1)$.

Durocher et al. [9] described an $O(n)$-space data structure that supports constant-time

range majority queries; this data structure is then extended to range α-majority queries, a generalization of range majority.

3 First Method: $O(\sqrt{n})$ Query Time and $O(n)$ Space

We begin by presenting a linear-space data structure with $O(\sqrt{n})$ query time, improving Krizanc et al.'s result [16] by a $\log\log$ factor. We build on the data structure of Krizanc et al. and introduce a different technique that avoids the need for predecessor search. We will actually establish the following time-space tradeoff—the linear-space result follows by setting the parameter $s = \lceil\sqrt{n}\rceil$.

▶ **Theorem 1.** *Given an array $A[1:n]$ and any fixed value $s \in [1,n]$, there exists a data structure requiring $O(n+s^2)$ space that supports range mode queries on A in $O(n/s)$ time.*

The following observation will be useful:

▶ **Lemma 2** (Krizanc et al. [16])**.** *Let A and B be any multisets. If c is a mode of $A \cup B$ and $c \notin A$, then c is a mode of B.*

Data Structure Precomputation. Given input array $A[1:n]$, let D denote the set of distinct elements stored in A and assume some arbitrary ordering on the elements. We first apply rank space reduction: construct an array $B[1:n]$ such that for each i, $B[i]$ stores the rank of $A[i]$ in D. Here, $B[i] \in \{1,\ldots,\Delta\}$. For any a, i, and j, $B[a]$ is a mode of $B[i:j]$ if and only if $A[a]$ is a mode of $A[i:j]$. For simplicity, we describe our data structures in terms of array B; a table look-up provides a direct bijective mapping from $\{1,\ldots,\Delta\}$ to D. Set D, array B, and the value Δ are independent of any query range and can be computed in $O(n\log\Delta)$ time during preprocessing.

For each $a \in \{1,\ldots,\Delta\}$, let $Q_a = \{b \mid B[b] = a\}$. That is, Q_a is the set of indices b such that $B[b] = a$. For any a, a range counting query for element a in $B[i:j]$ can be answered by searching for the predecessors of i and j, respectively, in the set Q_a; the difference of their indices is the frequency of a in $B[i:j]$ [16]. Such a range counting query can be implemented using an efficient predecessor data structure in $\Theta(\log\log n)$ time in the worst case (e.g., [22]).

The following related decision problem, however, can be answered in *constant* time by a linear-space data structure: does $B[i:j]$ contain at least q instances of element $B[i]$? This question can be answered by a "select" query that returns the index of the qth instance of $B[i]$ in $B[i:n]$. For each $a \in \{1,\ldots,\Delta\}$, store the set Q_a as an ordered array (also denoted Q_a for simplicity). Define a rank array $B'[1:n]$ such that for all b, $B'[b]$ denotes the rank (i.e., the index) of b in $Q_{B[b]}$. Given any q, i, and j, to determine whether $B[i:j]$ contains at least q instances of $B[i]$ it suffices to check whether $Q_{B[i]}[B'[i]+q-1] \le j$. Since array $Q_{B[i]}$ stores the sequence of indices of instances of element $B[i]$ in B, looking ahead $q-1$ positions in $Q_{B[i]}$ returns the index of the qth occurrence of element $B[i]$ in $B[i:n]$; if this index is at most j, then the frequency of $B[i]$ in $B[i:j]$ is at least q. If the index $B'[i]+q-1$ exceeds the size of the array $Q_{B[i]}$, then the query returns a negative answer. This gives the following lemma:

▶ **Lemma 3.** *Given an array $A[1:n]$, there exists a data structure requiring $O(n)$ space that can determine in constant time for any $0 \le i \le j \le n$ and any q whether $A[i:j]$ contains at least q instances of element $A[i]$.*

Following Krizanc et al. [16], given any $s \in [1,n]$ we partition array B into s blocks of size $t = \lceil n/s\rceil$. That is, for each $i \in \{0,\ldots,s-2\}$, the ith block spans $B[i\cdot t+1:(i+1)t]$

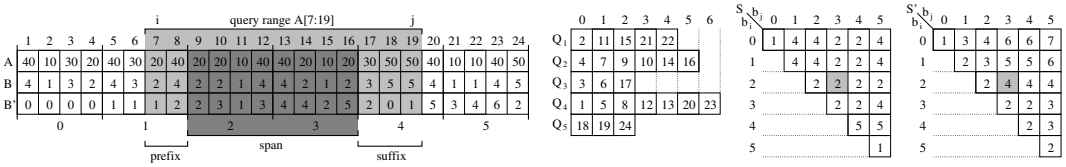


Figure 1 The array has size $n = 24$ (of which $\Delta = 5$ are distinct), partitioned into $s = 6$ blocks of size $t = 4$. The query range is $A[i : j] = A[7 : 19]$, for which the unique mode is 20, occurring with frequency 5. The corresponding mode of $B[i : j]$ is 2. The query range is partitioned into the prefix $B[7 : 8]$, the span $B[9 : 16]$, and the suffix $B[17 : 19]$. The span covers blocks $b_i = 2$ to $b_j = 3$, for which the corresponding mode is $S[2, 3] = 2$, occurring with frequency $S'[2, 3] = 4$.

and the last block spans $B[(s-1)t+1 : n]$. We precompute tables $S[0 : s-1, 0 : s-1]$ and $S'[0 : s-1, 0 : s-1]$, each of size $\Theta(s^2)$, such that for any $0 \le b_i \le b_j < s$, $S[b_i, b_j]$ stores a mode of $B[b_i t + 1 : (b_j + 1)t]$ and $S'[b_i, b_j]$ stores the corresponding frequency.

The arrays $Q_1, \ldots, Q_\Delta$ can be constructed in $O(n)$ total time in a single scan of array B. The arrays S and S' (which we call the *mode table*) can be constructed in $O(n \cdot s)$ time by scanning array B s times, computing one row of each array S and S' per scan. Thus, the total precomputation time required to initialize the data structure is $O(n \cdot s)$.

Query Algorithm. Given a query range $B[i : j]$, let $b_i = \lceil (i-1)/t \rceil$ and $b_j = \lfloor j/t \rfloor - 1$ denote the respective indices of the first and last blocks completely contained within $B[i : j]$. We refer to $B[b_i t + 1 : (b_j + 1)t]$ as the *span* of the query range, to $B[i : \min\{b_i t, j\}]$ as its *prefix*, and to $B[\max\{(b_j + 1)t + 1, i\} : j]$ as its *suffix*. One or more of the prefix, span, and suffix may be empty; in particular, if $b_i > b_j$, then the span is empty. See Figure 1.

The value $c = S[b_i, b_j]$ is a mode of the span with frequency $f_c = S'[b_i, b_j]$. If the span is empty, then let $f_c = 0$. By Lemma 2, either c is a mode of $B[i : j]$ or some element of the prefix or suffix is a mode of $B[i : j]$. Thus, to find a mode of $B[i : j]$, we verify for every element in the prefix and suffix whether its frequency in $B[i : j]$ exceeds f_c and, if so, we identify this element as a *candidate* mode and count its additional occurrences in $B[i : j]$.

We now describe how to compute the frequency of all candidate elements in the prefix, storing the value and frequency of the current best candidate in c and f_c; an analogous procedure is applied to the suffix. Sequentially scan the items in the prefix starting at the leftmost index, i, and let x denote the index of the current item. If $Q_{B[x]}[B'[x] - 1] \ge i$, then an instance of element $B[x]$ appears in $B[i : x-1]$, and its frequency has been counted already; in this case, simply skip $B[x]$ and increment x. Otherwise, check whether the frequency of $B[x]$ in $B[i : j]$ (which is equivalent to the frequency of $B[x]$ in $B[x : j]$) is at least f_c by Lemma 3 (i.e., by testing whether $Q_{B[x]}[B'[x] + f_c - 1] \le j$). If not, we again skip $B[x]$. Otherwise, $B[x]$ is a candidate, and the exact frequency of $B[x]$ in $B[i : j]$ can be counted by a linear scan[2] of $Q_{B[x]}$, starting at index $B'[x] + f_c - 1$ and terminating upon reaching either an index y such that $Q_{B[x]}[y] > j$ or the end of array $Q_{B[x]}$ (i.e., $y = |Q_{B[x]}| + 1$). That is, $Q_{B[x]}[y]$ denotes the index of the first instance of element $B[x]$ that lies beyond the query range $B[i : j]$ (or no such element exists). Consequently, the frequency of $B[x]$ in $B[i : j]$ is $f_x = y - B'[x]$. Update the current best candidate: $c \leftarrow B[x]$ and $f_c \leftarrow f_x$.

After all elements in the prefix and suffix have been processed, a mode of $B[i : j]$ and its frequency are stored in c and f_c, respectively.

[2] Although the time required to complete a linear scan could be reduced by instead using a binary search or a more efficient predecessor data structure, the worst-case time remains unchanged; for simplicity, a linear scan suffices.

Analysis. Excluding the linear scans of $Q_{B[x]}$, the query cost is clearly bounded by $O(t)$. For each candidate $B[x]$ encountered during the processing of the prefix, the cost of the linear scan of $Q_{B[x]}$ is $O(f_x - f_c)$. Since f_c is at least the frequency of the mode of the span, at least $f_x - f_c$ instances of $B[x]$ must occur in the prefix or suffix. We can thus charge the cost of the scan to these instances. Since each element $B[x]$ is considered a candidate at most once (during its first appearance) in the prefix, we conclude that the total cost of all the linear scans is proportional to the total number of elements in the prefix, i.e., $O(t)$. An analogous argument holds for the cost of processing the suffix. Therefore, a range mode query requires $O(t) = O(n/s)$ total time. The data structure requires $O(n)$ space to store the arrays A, B, and B', $O(n)$ total space to store the arrays $Q_1, \ldots, Q_\Delta$, and $O(s^2)$ space to store the tables S and S'. This proves Theorem 1.

4 Second Method: $O(\sqrt{n/w})$ Query Time and $O(n)$ Space When $m \leq \sqrt{nw}$

Our second method is a refinement of the first method (from Section 3), in which we store the mode table (S and S') more compactly by an encoding scheme that enables efficient retrieval of the relevant information, using techniques from succinct data structures, specifically, for *rank/select* operations. We show how to reduce a query to four rank/select operations. These new ideas allow us to improve the space bound in Theorem 1 by a factor of w, which enables us to use a slightly larger number of blocks, s, which in turn leads to an improved query time. However, there is one important caveat: our space-saving technique only works when the maximum frequency is small, namely, when $m \leq s$. Specifically, we will prove the following theorem in this section: choosing $s = \lceil \sqrt{nw} \rceil$ gives $O(n)$ space and $O(\sqrt{n/w})$ query time for $m \leq \sqrt{nw}$.

▶ **Theorem 4.** *Given an array $A[1:n]$ and any fixed $s \in [1, n]$ such that $m \leq s$ (where m is the frequency of the overall mode), there exists a data structure requiring $O(n + s^2/w)$ space that supports range mode queries on A in $O(n/s)$ time.*

Modified Data Structure. Recall that for a span from block b_i to block b_j, the mode table stores a mode of the span and its frequency in $S[b_i, b_j]$ and $S'[b_i, b_j]$, respectively. As we will show, a mode of the span can be computed efficiently if its frequency is known; consequently, we omit table S. Also, instead of storing the frequency of the mode explicitly, we store column-to-column frequency deltas (i.e., differences of adjacent frequency values); observe that frequency values are monotone increasing across each row. We encode the frequency deltas for a single row as a bit string, where a zero bit represents an increment in the frequency of the mode (i.e., each frequency delta is encoded in unary) and a one bit represents a former cell boundary. In any row, the number of ones is at most the number of blocks, s, and the number of zeroes is at most $m \leq s$. Precompute a data structure that uses a linear number of bits to support $O(1)$-time binary *rank* and *select* operations on each row (e.g., see [18]):[3] given a binary string, for each $a \in \{0, 1\}$, $\text{rank}_a(i)$ returns the number of times a occurs in the first i positions of the string, and $\text{select}_a(i)$ returns the position of the ith occurrence of a in the string. Thus, each row of the table uses $O(s)$ bits of space. The table has s rows and requires $O(s^2)$ bits of space in total. We pack these bits into words, resulting in an $O(s^2/w)$-space data structure.

[3] Succinct data structures can ensure that space usage is very close to the length of the bit string up to lower-order terms, but this fact is not needed in our application.

Modified Query Algorithm. Assuming we know a mode of the span and its frequency, we can process the prefix and suffix ranges in $O(t)$ time as before. Our attention turns now to determining a mode of the span and its frequency. We first obtain the frequency of the mode of the span in $O(1)$ time using rank and select queries on the bit string of the b_ith row:

$$pos_{b_j} \leftarrow \text{select}_1(b_j - b_i + 1), \qquad \text{and} \qquad freq \leftarrow \text{rank}_0(pos_{b_j}).$$

Having found the frequency of the mode, identifying a mode itself is still a tricky problem. We proceed in two steps. We first determine the block in which the last occurrence of a mode lies, in $O(1)$ time, as follows:

$$pos_{\text{last}} \leftarrow \text{select}_0(freq), \qquad \text{and} \qquad b_{\text{last}} \leftarrow \text{rank}_1(pos_{\text{last}}) + b_i.$$

Next we find a mode of the span by iteratively examining each element in block b_{last}, using a technique analogous to that for processing a suffix from Section 3. By Lemma 3 (reversed with $j \leq i$), we can check whether each element $B[x]$ in b_{last} has frequency *freq* in $B[b_i t + 1 : x]$, in $O(1)$ time per element. If the mode occurs multiple times in block b_{last}, its last occurrence will be successfully identified. Processing block b_{last} requires $O(t)$ total time. We conclude that the total query time is $O(t) = O(n/s)$ time. This proves Theorem 4.

5 Third Method: $O(\Delta)$ Query Time and $O(n)$ Space

In this section, we take a quick detour and consider a third method that has query time sensitive to Δ, the number of distinct elements; this "detour" turns out to be essential in assembling our final solution. We show the following:

▶ **Theorem 5.** *Given an array $A[1 : n]$, there exists a data structure requiring $O(n)$ space that supports range mode queries on A in $O(\Delta)$ time, where Δ denotes the number of distinct elements in A.*

The proof is simple: to answer a range mode query, the approach is to compute the frequency (in the query range) for each of the Δ possible elements explicitly, and then just compute the maximum in $O(\Delta)$ time.

Data Structure Precomputation. As before, we work with the array B by rank space reduction. This time, we divide B into blocks of size $t = \Delta$. For each $i \in \{1, \ldots, \lfloor n/\Delta \rfloor\}$, and for every $x \in \{1, \ldots, \Delta\}$, store the frequency $C_i[x]$ of x in the range $B[1 : i\Delta]$. The total size of all these *frequency tables* is $O((n/\Delta) \cdot \Delta) = O(n)$. The preprocessing time required is $O(n)$ (or $O(n \log \Delta)$ time if Δ or B must be computed).

Query Algorithm. Given a query range $B[i : j]$, as mentioned, it suffices to compute the frequency of x in $B[i : j]$ for every $x \in \{1, \ldots, \Delta\}$.

Let $b_j = \lfloor j/\Delta \rfloor - 1$. We can compute the frequency $C(x)$ of x in the suffix $B[b_j\Delta + 1 : j]$ for every $x \in \{1, \ldots, \Delta\}$ by a linear scan, in $O(\Delta)$ time since the suffix has size at most Δ. Then the frequency of x in $B[1, j]$ is given by $C_{b_j}[x] + C(x)$. The frequency of x in $B[1, i]$ can be computed similarly. The frequency of x in $B[i, j]$ is just the difference of these two numbers. The total query time is clearly $O(\Delta)$. This proves Theorem 5.

6 Final Method: $O(\sqrt{n/w})$ Query Time and $O(n)$ Space

We are finally ready to present our improved linear-space data structure with $O(\sqrt{n/w})$ query time. Our final idea is simple: if the elements all have small frequencies, the second

method (Section 4) already works well; otherwise, the number of distinct elements with large frequencies is small, and so the third method (Section 5) can be applied instead.

More precisely, let s be any fixed value in $[1, n]$. Partition the elements of A into those with *low* frequencies, i.e., at most s, and those with *high* frequencies, i.e., greater than s. A mode of the low-frequency elements has frequency at most s. Thus we can apply Theorem 4 to build an $O(n + s^2/w)$-space range mode query data structure on the low-frequency elements to support $O(n/s)$ query time. On the other hand, there are at most n/s distinct high-frequency elements. Thus we can apply Theorem 5 to build an $O(n)$-space range mode query data structure on the high-frequency elements to support $O(n/s)$ query time. The following simple decomposition lemma allows us to combine the two structures:

▶ **Lemma 6.** *Given an array $A[1 : n]$ and any ordered partition of A into two arrays $B_1[1 : n']$ and $B_2[1 : n - n']$ such that no element in B_1 occurs in B_2 nor vice versa, if there exist respective $s_1(n)$- and $s_2(n)$-space data structures that support range mode queries on B_1 and B_2 in $t_1(n)$ and $t_2(n)$ time, then there exists an $O(n + s_1(n) + s_2(n))$-space data structure that supports range mode query on A in $O(t_1(n) + t_2(n))$ time.*

Proof. For each $a \in \{1, 2\}$ and $i \in \{1, \ldots, n\}$, precompute $I_a[i]$, the index in the B_a array of the first element in A to the right of $A[i]$ that lies in B_a; and precompute $J_a[i]$, the index in the B_a array of the first element in A to the left of $A[i]$ that lies in B_a. Given a range query $A[i : j]$, we can compute the mode in the range $B_1[I_1[i], J_1[j]]$ and the mode in the range $B_2[I_2[i], J_2[j]]$ and determine which has larger frequency; this is a mode of $A[i : j]$. ◀

We have thus completed the proof of our main theorem:

▶ **Theorem 7.** *Given an array $A[1 : n]$ and any fixed $s \in [1, n]$, there exists a data structure requiring $O(n + s^2/w)$ space that supports range mode queries on A in $O(n/s)$ time. In particular, by setting $s = \lceil \sqrt{nw} \rceil$, there exists a data structure requiring $O(n)$ space that supports range mode queries on A in $O(\sqrt{n/w})$ time.*

7 Boolean Matrix Multiplication and Range Mode

In this section, we show that boolean matrix multiplication of two $\sqrt{n} \times \sqrt{n}$ matrices reduces to n range mode queries in an array of size $O(n)$. Greve et al. [12] observe the following:

▶ Observation 8 (Greve et al. [12]). Let S be a multiset whose elements belong to a universe U. Adding one of each element in U to S increases the frequency of the mode of S by one.

▶ Observation 9 (Greve et al. [12]). Let S_1 and S_2 be two sets (not multisets) and let S be the multiset union of S_1 and S_2. The frequency of the mode of S is one if $S_1 \cap S_2 = \emptyset$ and it is two if $S_1 \cap S_2 \neq \emptyset$.

Now let A and B be two $\sqrt{n} \times \sqrt{n}$ boolean matrices for which we are to compute the product $C = A \cdot B$. The entry $c_{i,j}$ in C must be 1 precisely if there exists at least one index k, where $1 \leq k \leq \sqrt{n}$, such that $a_{i,k} = b_{k,j} = 1$. Our goal is to determine whether this is the case using one range mode query for each entry $c_{i,j}$. Our first step in achieving this is to transform each row of A and each column of B into a set. For the ith row of A, we construct the set A_i containing all those indices k for which $a_{i,k} = 1$, i.e., $A_i = \{k \mid a_{i,k} = 1\}$. Similarly we let $B_j = \{k \mid b_{k,j} = 1\}$. Clearly $c_{i,j} = 1$ if and only if $A_i \cap B_j \neq \emptyset$. By Observation 9, this can be tested if we can determine the frequency of the mode in the multiset union of A_i and B_j. Our last step is thus to embed all the sets A_i and B_j into an array, such that we can use range mode queries to perform these intersection tests for every pair i, j. Our

constructed array M has two parts, a left part L and a right part R. The array M is then simply the concatenation of L and R. The array L represents all the sets A_i. It consists of $\sqrt{n}$ blocks of $\sqrt{n}$ entries. The ith block (entries $L[(i-1)\sqrt{n}+1 : i\sqrt{n}]$) represents the set A_i, and it consists of the elements $\{1,\ldots,\sqrt{n}\} \setminus A_i$ in some arbitrary order, followed by the elements of A_i in some arbitrary order. The array R similarly represents the sets B_j and it also consists of $\sqrt{n}$ blocks of $\sqrt{n}$ entries. The jth block represents the set B_j and it consists of the elements in B_j in some arbitrary order, followed by the elements $\{1,\ldots,\sqrt{n}\} \setminus B_j$ in some arbitrary order.

Now assume that $|A_i|$ and $|B_j|$ are known for each set A_i and B_j. We can now determine whether $A_i \cap B_j \neq \emptyset$ (i.e., whether $c_{i,j} = 1$) from the result of the range mode query on $M[\text{start}(i) : \text{end}(j)]$, where

$$\text{start}(i) = (i-1)\sqrt{n} + 1 + \sqrt{n} - |A_i| \qquad \text{and} \qquad \text{end}(j) = n + (j-1)\sqrt{n} + |B_j|.$$

To see this, first observe that $\text{start}(i)$ is the first index in M of the elements in A_i, and that $\text{end}(j)$ is the last index in M of the elements in B_j. In addition to a suffix of the block representing A_i and a prefix of the block representing B_j, the subarray $M[\text{start}(i) : \text{end}(j)]$ contains $\sqrt{n} - i$ complete blocks from L and $j-1$ complete blocks from R. Since a complete block contains all the elements $\{1,\ldots,\sqrt{n}\}$, it follows from Observations 8 and 9 that $A_i \cap B_j \neq \emptyset$ (i.e., $c_{i,j} = 1$) if and only if the frequency of the mode in $M[\text{start}(i), \text{end}(j)]$ is $2 + \sqrt{n} - i + j - 1$. The answer to the query $(\text{start}(i), \text{end}(j))$ thus allows us to determine whether $c_{i,j} = 1$ or 0. The array M and the values $|A_i|$ and $|B_j|$ can clearly be computed in linear time when given matrices A and B, thus we have the following result:

▶ **Theorem 10.** *Let $p(n)$ be the preprocessing time of a range mode data structure and $q(n)$ its query time. Then boolean matrix multiplication on two $\sqrt{n} \times \sqrt{n}$ matrices can be solved in time $O(p(n) + n \cdot q(n) + n)$.*

8 Higher Dimensions

We now consider generalizations of the range mode problem to Euclidean spaces of constant dimension d. Given a set P of n points in $\mathbb{R}^d$, each of which is assigned a colour, we consider the problem of constructing an efficient data structure to support queries that return a most frequently occurring colour in $P \cap Q$ for a query range $Q \subseteq \mathbb{R}^d$. We consider orthogonal range queries in Section 8.1 and halfspace range queries in Section 8.2.

8.1 Orthogonal Ranges

We generalize the technique of Krizanc et al. [16] by dividing space into s^d grid cells such that there are $O(n/s)$ points between any two consecutive parallel grid hyperplanes. The generalization of a span of a query range Q is the largest rectangle inside Q whose sides lie along grid hyperplanes. There are s^{2d} distinct spans and for each we precompute and store the mode of the span. This component of our data structure thus requires $O(s^{2d})$ space. For each set of points of a given colour, we also build an orthogonal range counting data structure [8] with polylogarithmic space overhead that answers queries in polylogarithmic time (see [13] for the best known solution, using $O(n(\log n/\log\log n)^{d-2})$ space and with $O((\log n/\log\log n)^{d-1})$ query time). Across all colours, these data structures require $O(n \operatorname{polylog} n)$ space.

Given a query hyperrectangle Q we use binary search amongst the grid hyperplanes in order to determine the slabs in which Q's sides lie. We then determine the mode of Q's span

in $O(1)$ time from our precomputed table. For each of the $2d$ sides of Q we must additionally consider each of the $O(n/s)$ points in the slab in which the side lies. For each such point, we count the number of points of its colour in Q using the range counting data structure of its colour in polylogarithmic time to find the actual mode. So, the running time of a query is $O\left(2d \cdot (n/s) \cdot \operatorname{polylog} n\right) = O\left((n/s) \cdot \operatorname{polylog} n\right)$ time.

▶ **Theorem 11.** *Given a set P of n points in $\mathbb{R}^d$, each of which is assigned a colour, and any fixed $s \in \{1, \ldots, n\}$, there exists a data structure requiring $O(n \operatorname{polylog} n + s^{2d})$ space that supports orthogonal range mode queries in $O\left((n/s) \cdot \operatorname{polylog} n\right)$ time. In particular, by setting $s = \lceil n^{1/2d} \rceil$, there exists a data structure requiring $O(n \operatorname{polylog} n)$ space that supports range mode queries in $O(n^{1-1/2d} \operatorname{polylog} n)$ time.*

Alternatively, we can guarantee $O(n)$ space if we increase the query time by an n^ϵ factor, by switching to a linear-space data structure for orthogonal range counting with $O(n^\epsilon)$ query time (by using a range tree [8] with n^ϵ fan-out).

8.2 Halfspace Ranges

We now consider halfspace range queries. We work in *dual* space [8], where the input is transformed into n hyperplanes, each assigned a colour, and a query halfspace is transformed into a point. A query for a dual point q returns the most frequently occurring colour amongst the hyperplanes that lie below q. Let $s \in \{1, \ldots, n\}$ be a fixed parameter specified by the user. We use the key concept of cuttings [7] from computational geometry. Given a set of n hyperplanes in $\mathbb{R}^d$, a $(1/r)$-*cutting* is a partition of $\mathbb{R}^d$ into simplicial cells such that each cell intersects at most n/r hyperplanes. The following is known [7, 6]:

▶ **Lemma 12.** *For any set of n hyperplanes in $\mathbb{R}^d$, there exists a $(1/r)$-cutting with $O(r^d)$ cells. Furthermore, there is a data structure for point location in the $(1/r)$-cutting, also requiring $O(r^d)$ space and answering queries in $O(\log r)$ time.*

We set $r = (n \cdot s^{d-1})^{1/d}$. For each cell γ in the cutting, we store the mode of the hyperplanes that lie strictly below γ. This component of our data structure requires $O(r^d) = O(n \cdot s^{d-1})$ space. In primal space, we build a simplex range reporting data structure [4, 17] for all of the points with $S = O(n \cdot s^{d-1})$ space. This data structure reports the k points in a query simplex in $O((n/S^{1/d}) \operatorname{polylog} n + k) = O((n/s)^{1-1/d} \operatorname{polylog} n + k)$ time. Also, for each colour i, we build a separate halfspace range counting data structure [4, 17] for the n_i points of colour i, with $S_i = O(n_i \cdot s^{d-1})$ space and $O(n_i/S_i^{1/d} + \log n_i) = O((n_i/s)^{1-1/d} + \log n)$ query time. The total space is $O(n \cdot s^{d-1})$.

Given a dual query point q, we first identify the cell γ of the $(1/r)$-cutting that contains q in $O(\log r)$ time. The mode of the hyperplanes below q is either the colour stored at cell γ or one of the colours of the hyperplanes intersecting γ. We can find the $O(n/r)$ hyperplanes intersecting γ by simplex range reporting in primal space in $O((n/s)^{1-1/d} \operatorname{polylog} n + n/r)$ time, since the set of all hyperplanes intersecting a simplex dualizes to a polyhedron of $O(1)$ size. For each hyperplane that intersects γ and lies below q, we perform a halfspace range counting query for the points of the colour of the hyperplane in primal space to determine the actual mode. The running time of this step is $O\left(\sum_{i=1}^{O(n/r)} (n_i/s)^{1-1/d} + (n/r) \log n\right)$. By Hölder's inequality, the sum in the first term is bounded by $O((n/r)^{1/d} \cdot (n/s)^{1-1/d}) = O((n/s)^{1-1/d^2})$ for $r = (n \cdot s^{d-1})^{1/d}$. The second term $(n/r) \log n = (n/s)^{1-1/d} \log n$ does not dominate except when $n/s = O(\operatorname{polylog} n)$.

▶ **Theorem 13.** *Given a set P of n points in $\mathbb{R}^d$, each of which is assigned a colour, and any fixed $s \in \{1, \ldots, n\}$, there exists a data structure requiring $O(n \cdot s^{d-1})$ space that supports halfspace range mode queries in $O((n/s)^{1-1/d^2} + \operatorname{polylog} n)$ time. In particular, by setting $s = 1$, there exists a data structure requiring $O(n)$ space that supports halfspace range mode queries in $O(n^{1-1/d^2})$ time.*

A similar approach works for other ranges (e.g., simplices, balls, and other constant-degree semialgebraic sets) by transforming query ranges to query points in a higher dimension, and using cuttings in this higher-dimensional space.

9 Discussion and Directions for Future Research

We close by mentioning a few interesting open problems. A useful generalization of the problem is to return the kth most frequently occurring element (or the k most frequent elements) in a query range. Due to its dependence on precomputed modes stored in array S, an analogous generalization of our methods (except for the third method) seems unlikely without a significant increase in space, if k is large.

▶ Open Problem 1. Construct an $O(n)$-space data structure for identifying the kth most frequently occurring element (or the k most frequent elements) in the range $A[i : j]$ in time $O(n^{1-\epsilon})$ (or $O(n^{1-\epsilon} + k)$) for some constant $\epsilon > 0$, where i, j, and k are given at query time.

We have given (near-)linear-space data structures for multiple variants of range mode, including orthogonal range mode for a d-dimensional point set and halfspace range mode for a d-dimensional point set. Our results, in various ways, build on and generalize the techniques of Krizanc et al. [16]. It is unknown whether there are entirely different approaches that can achieve smaller exponents on n in the query times.

▶ Open Problem 2. Is there a linear-space dynamic data structure for range mode in an array that supports queries and updates in $O(\sqrt{n} \operatorname{polylog} n)$ time? In the full paper we give respective dynamic data structures with $O(n)$ space and $O(n^{3/4} \operatorname{polylog} n)$ query and update times, and with $O(n^{4/3})$ space and $O(n^{2/3} \operatorname{polylog} n)$ query and update times.

▶ Open Problem 3. Is there a (near-)linear-space data structure for orthogonal range mode in $\mathbb{R}^d$ that supports queries in $o(n^{1-1/2d})$ time?

▶ Open Problem 4. Is there a linear-space data structure for halfspace range mode in $\mathbb{R}^d$ that supports queries in $o(n^{1-1/d^2})$ time?

Lastly, the following open problem is likely difficult since currently no techniques seem capable of proving unconditional super-polylogarithmic cell probe lower bounds:

▶ Open Problem 5. Prove a tight, unconditional lower bound on the worst-case query time required by any $O(n)$-space data structure that supports range mode queries on an array of n items.

Acknowledgements. The authors thank Peyman Afshani, Francisco Claude, Meng He, Ian Munro, Patrick Nicholson, Matthew Skala, and Norbert Zeh for discussing various topics related to range searching.

References

1 N. Bansal and R. Williams. Regularity lemmas and combinatorial algorithms. In *Proc. IEEE FOCS*, pages 745–754, 2009.

2 P. Bose, E. Kranakis, P. Morin, and Y. Tang. Approximate range mode and range median queries. In *Proc. STACS*, volume 3404 of *LNCS*, pages 377–388. Springer, 2005.

3 G. S. Brodal, B. Gfeller, A. G. Jørgensen, and P. Sanders. Towards optimal range medians. *Theor. Comp. Sci.*, 412(24):2588–2601, 2011.

4 T. M. Chan. Optimal partition trees. In *Proc. ACM SoCG*, pages 1–10, 2010.

5 T. M. Chan and M. Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proc. ACM-SIAM SODA*, pages 161–173, 2010.

6 B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Disc. Comp. Geom.*, 9(2):145–158, 1993.

7 B. Chazelle. Cuttings. In *Handbook of Data Structures and Applications*, pages 25.1–25.10. CRC Press, 2005.

8 M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Heidelberg, Germany, 3rd edition, 2008.

9 S. Durocher, M. He, J. I. Munro, P. K. Nicholson, and M. Skala. Range majority in constant time and linear space. In *Proc. ICALP*, volume 6755 of *LNCS*, pages 244–255. Springer, 2011.

10 T. Gagie, S. J. Puglisi, and A. Turpin. Range quantile queries: Another virtue of wavelet trees. In *Proc. SPIRE*, volume 5721 of *LNCS*, pages 1–6. Springer, 2009.

11 B. Gfeller and P. Sanders. Towards optimal range medians. In *Proc. ICALP*, volume 5555 of *LNCS*, pages 475–486. Springer, 2009.

12 M. Greve, A. G. Jørgensen, K. D. Larsen, and J. Truelsen. Cell probe lower bounds and approximations for range mode. In *Proc. ICALP*, volume 6198 of *LNCS*, pages 605–616. Springer, 2010.

13 J. JáJá, C. W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proc. ISAAC*, volume 3341 of *LNCS*, pages 558–568. Springer, 2004.

14 A. G. Jørgensen and K. D. Larsen. Range selection and median: Tight cell probe lower bounds and adaptive data structures. In *Proc. ACM-SIAM SODA*, pages 805–813, 2011.

15 D. Krizanc, P. Morin, and M. Smid. Range mode and range median queries on lists and trees. In *Proc. ISAAC*, volume 2906 of *LNCS*, pages 517–526. Springer, 2003.

16 D. Krizanc, P. Morin, and M. Smid. Range mode and range median queries on lists and trees. *Nordic Journal of Computing*, 12:1–17, 2005.

17 J. Matoušek. Range searching with efficient hierarchical cuttings. *Disc. Comp. Geom.*, 10(2):157–182, 1993.

18 J. I. Munro. Tables. In V. Chandru and V. Vinay, editors, *Foundations of Software Technology and Theoretical Computer Science*, volume 1180 of *LNCS*, pages 37–42. Springer, 1996.

19 J. I. Munro and M. Spira. Sorting and searching in multisets. *SIAM J. Comp.*, 5(1):1–8, 1976.

20 H. Petersen. Improved bounds for range mode and range median queries. In *Proc. SOFSEM*, volume 4910 of *LNCS*, pages 418–423. Springer, 2008.

21 H. Petersen and S. Grabowski. Range mode and range median queries in constant time and sub-quadratic space. *Inf. Proc. Let.*, 109:225–228, 2009.

22 P. van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Inf. Proc. Let.*, 6(3):80–82, 1977.

23 V. Vassilevska Williams. Breaking the Coppersmith-Winograd barrier. http://www.cs.berkeley.edu/˜virgi/matrixmult.pdf, 2011.

Log-supermodular functions, functional clones and counting CSPs*

Andrei A. Bulatov[1], Martin Dyer[2], Leslie Ann Goldberg[3], and Mark Jerrum[4]

1 **School of Computing Science, Simon Fraser University**
Burnaby, Canada
abulatov@cs.sfu.edu
2 **School of Computing, University of Leeds**
Leeds LS2 9JT, UK
dyer@comp.leeds.ac.uk
3 **Department of Computer Science, University of Liverpool**
Liverpool, L69 3BX, UK
L.A.Goldberg@liverpool.ac.uk
4 **School of Mathematical Sciences, Queen Mary, University of London**
London E1 4NS, UK
m.jerrum@qmul.ac.uk

Abstract

Motivated by a desire to understand the computational complexity of counting constraint satisfaction problems (counting CSPs), particularly the complexity of approximation, we study functional clones of functions on the Boolean domain, which are analogous to the familiar relational clones constituting Post's lattice. One of these clones is the collection of log-supermodular (lsm) functions, which turns out to play a significant role in classifying counting CSPs. In our study, we assume that non-negative unary functions (weights) are available. Given this, we prove that there are no functional clones lying strictly between the clone of lsm functions and the total clone (containing all functions). Thus, any counting CSP that contains a single nontrivial non-lsm function is computationally as hard as any problem in #P. Furthermore, any non-trivial functional clone (in a sense that will be made precise below) contains the binary function "implies". As a consequence, all non-trivial counting CSPs (with non-negative unary weights assumed to be available) are computationally at least as difficult as #BIS, the problem of counting independent sets in a bipartite graph. There is empirical evidence that #BIS is hard to solve, even approximately.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes

Keywords and phrases counting constraint satisfaction problems, approximation, complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.302

1 Introduction

In the classical setting, a constraint satisfaction problem $\mathsf{CSP}(\Gamma)$ is specified by a finite domain D and constraint language Γ, which is a set of relations of varying arities over D. An instance of $\mathsf{CSP}(\Gamma)$ is a set of n variables taking values in D, together with a set of

* The work described in this paper was partly supported by EPSRC Research Grant (refs EP/I011528/1, EP/I011935/1 and EP/I012087/1) "Computational Counting", and by NSERC Discovery Grant. Part of the work was supported by a visit to the Isaac Newton Institute for Mathematical Sciences, as part of the programme "Discrete Analysis".

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 302–313

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

constraints on those variables. Each constraint is an a-ary relation R from Γ applied to an a-tuple of variables, the scope of the constraint. Thus, constraint satisfaction problems (CSPs) may be viewed as generalised satisfiability problems, among which usual satisfiability is a very special case.

The relational clone $\langle\Gamma\rangle_{\mathrm{R}}$ generated by a set Γ of relations is the set of relations that are expressible, in some precise sense termed "pp-definability", in terms of the base relations Γ. It turns out that if two sets of relations Γ and Γ' generate the same relational clone $\langle\Gamma\rangle_{\mathrm{R}} = \langle\Gamma'\rangle_{\mathrm{R}}$, then the computational complexity of the corresponding CSPs, $\mathsf{CSP}(\Gamma)$ and $\mathsf{CSP}(\Gamma')$, are the same. Relational clones have played a key role in the development of the complexity theory of CSPs: instead of considering all sets of relations Γ, one only needs to consider the ones that are relational clones. For an introduction to the algebraic theory of relational clones, see, for example, the expository chapter of Cohen and Jeavons [7].

Recently, there has been considerable interest in the computational complexity of counting CSPs. Here, the goal is to count the number of solutions rather than merely to decide if one exists. In fact, in order to encompass the computation of partition functions of models from statistical physics and other generating functions, it is reasonable to consider weighted sums, which can be expressed by replacing the relations in the constraint language by real- or complex-valued functions. Then the weight of an assignment is the product of the function values corresponding to that assignment, while the value of the CSP instance itself is the sum of the weights of all assignments. If I is an instance of such a counting CSP then we denote this weighted sum by $Z(I)$, and call it the "partition function of I" by analogy with the concept in statistical physics. For a finite set of functions Γ we are interested in the problem $\#\mathsf{CSP}(\Gamma)$: given an instance I using only functions from Γ, output $Z(I)$.

Our first goal (see §2) is to answer the question: what is the analogue of pp-definability, and hence of relational clones, in the context of (weighted) counting CSPs (#CSPs), and what insight does it provide into the computational complexity of these problems? At a high level, the answer to the first question is clear. View the relations in Γ as predicates. A relation is pp-definable over Γ in the classical sense if it can be expressed as the projection of a conjunction of predicates in Γ. (Projection is the operation of existential quantification over a certain subset of variables.) In order to adapt this concept to the counting setting, we should replace a conjunction of relations by a product of functions, and replace existential quantification (projection) by summation. However, in defining a counting analogue of pp-definability, a number of detailed decisions have to be made, and a number of delicate issues faced.

We call our proposed analogue of pp-definability "pps_ω-definability", and our analogue of relational clone "functional clone". There is at least one proposal in the literature for extending pp-definability to the algebraic/functional setting, that of Yamakami [18]. However, pps_ω-definability is more liberal than the corresponding notion in [18], and leads to a more inclusive functional clone. Our notion of pps_ω-definability includes a limiting operation. Without this limit, a functional clone may contain arbitrarily close approximations to a function F of interest, without including F itself.

Aside from a desire for tidiness, there is a good empirical motivation for introducing limits. Just as pp-definability is closely related to polynomial-time reductions between classical CSPs, so is pps_ω-definability related to approximation-preserving reductions between counting CSPs. (Lemma 9 is a precise statement of this connection.) Many approximation-preserving reductions in the literature (for example, [12]) are based not on a fixed "gadget" but on sequences of increasingly-large gadgets that come arbitrarily close to some property without actually attaining it. Our notion of pps_ω-definability seems exactly to capture this phenomenon.

Our second, more concrete goal (see §3–§5) is to explore the role of log-supermodular functions in the classification of functional clones, and hence in the complexity of approximating #CSPs. We restrict attention to the Boolean situation; that is, the domain is $\{0,1\}$ and the allowed functions are of the form $\{0,1\}^k \to \mathbb{R}^{\geq 0}$ for some integer k. A function with Boolean domain is said to be log-supermodular if the logarithm of it is supermodular. It is a non-trivial fact (Lemma 5) that the set LSM of log-supermodular functions is in fact a functional clone. We examine the landscape of functional clones under the assumption that non-negative unary functions (weights) are available. (Such an assumption is quite usual in related work, such as Cai, Lu and Xia's work on classifying "Holant*" problems [6].) Adding non-negative weights makes the classification of functional clones more tractable, though we are still unable to provide a complete inventory. On the other hand, adding all unary weights leads to a less rich (and more pessimistic) landscape [18]: negative weights introduce cancellation, which tends to drive approximate counting CSPs in the direction of intractability.

One particularly simple functional clone is the one generated by disequality. (Following convention, we allow equality for free, in addition to the non-negative weights mentioned earlier.) A counting CSP derived from this clone is trivial to solve exactly, as the partition function factorises. Let us say that functions from this clone are of "product form". Our main result (Theorem 8) is that any clone that contains a function F that is not of product form necessarily contains IMP, the binary (i.e., arity-2) function that takes the value 1, unless its first argument is 1 and its second is 0, when it takes the value 0. (The complexity-theoretic consequence of this will be discussed presently.) Furthermore (also Theorem 8), if F is not log-supermodular (and is not in the clone generated by disequality), then the clone contains all functions. Note that a large part of the functional clone landscape — below the clone generated by IMP and above LSM — is very simple. If there is a complex landscape of functional clones it must lie between the functional clone generated by IMP and the class of functions LSM.

We present also an efficient version of pps_ω-definability, and a corresponding notion of functional clone, that allows complexity-theoretical consequences to be deduced (Theorem 10). This is the third contribution of the paper (see §6). The last three authors, together with Greenhill [10], studied the complexity of counting problems expressible using IMP. They identified a class of natural problems of this form (which has since grown considerably) which are interreducible via approximation-preserving reduction, and for which no efficient approximation algorithm (FPRAS) is known. They conjectured that problems in this class do not admit an FPRAS. If this is so then $\#\mathsf{CSP}(\mathcal{F})$ is computationally intractable (in the presence of nonnegative weights) whenever $\mathcal{F}$ contains a function F that is not of product form. Furthermore, if F is not log-supermodular, then the counting problem $\#\mathsf{CSP}(\mathcal{F})$ is universal for Boolean counting CSPs and hence is provably NP-hard to approximate.

Although we focus on approximation of the partition functions of (weighted) #CSPs in this paper, there is of course an extensive literature on exact computation; see, e.g., Cai, Chen and Lu [5] and prior work.

2 Functional clones

Let $(R,+,\times)$ be any subsemiring of $(\mathbb{C},+,\times)$, where $\mathbb{C}$ denotes the complex numbers, and D a finite domain. For $n \in \mathbb{N}$, denote by $\mathcal{U}_n$ the set of all functions $D^n \to R$; also denote by $\mathcal{U} = \mathcal{U}_0 \cup \mathcal{U}_1 \cup \mathcal{U}_2 \cup \cdots$ the set of functions of all arities. Suppose $\mathcal{F} \subseteq \mathcal{U}$ is some collection of functions, $V = \{v_1, \ldots, v_n\}$ is a set of variables and $\boldsymbol{x} : \{v_1, \ldots, v_n\} \to D$

is an assignment to those variables. An atomic formula has the form $\varphi = G(v_{i_1}, \ldots, v_{i_a})$ where $G \in \mathcal{F}$, $a = a(G)$ is the arity of G, and $(v_{i_1}, v_{i_2}, \ldots, v_{i_a}) \in V^a$ is a scope. Note that repeated variables are allowed. The function $F_\varphi : D^n \to R$ represented by the atomic formula $\varphi = G(v_{i_1}, \ldots, v_{i_a})$ is just $F_\varphi(\boldsymbol{x}) = G(\boldsymbol{x}(v_{i_1}), \ldots, \boldsymbol{x}(v_{i_a})) = G(x_{i_1}, \ldots, x_{i_a})$, where from now on we write $x_j = \boldsymbol{x}(v_j)$.

A pps-formula ("primitive product summation formula") is a summation of a product of atomic formulas. A pps-formula ψ over $\mathcal{F}$ in variables $V' = \{v_1, \ldots, v_{n+m}\}$ has the form $\psi = \sum_{v_{n+1}, \ldots, v_{n+m}} \prod_{j=1}^{s} \varphi_j$, where φ_j are all atomic formulas over $\mathcal{F}$ in the variables V'. (The variables V are free, and the others, $V' \setminus V$, are bound.) The formula ψ specifies a function $F_\psi : D^n \to R$ in the following way:

$$F_\psi(\boldsymbol{x}) = \sum_{\boldsymbol{y} \in D^m} \prod_{j=1}^{s} F_{\varphi_j}(\boldsymbol{x}, \boldsymbol{y}), \tag{1}$$

where $\boldsymbol{x}$ and $\boldsymbol{y}$ are assignments $\boldsymbol{x} : \{v_1, \ldots, v_n\} \to D$ and $\boldsymbol{y} : \{v_{n+1}, \ldots, v_{n+m}\} \to D$. The functional clone $\langle \mathcal{F} \rangle$ generated by $\mathcal{F}$ is the set of all functions in $\mathcal{U}$ that can be represented by a pps-formula over $\mathcal{F} \cup \{\mathrm{EQ}\}$ where EQ is the binary equality function defined by $\mathrm{EQ}(x, x) = 1$ and $\mathrm{EQ}(x, y) = 0$ for $x \neq y$. We refer to the pps-formula as an "implementation" of the function. We use the following lemma.

▶ **Lemma 1.** *If $G \in \langle \mathcal{F} \rangle$ then $\langle \mathcal{F}, G \rangle = \langle \mathcal{F} \rangle$.*

To make the next step we suppose that R is dense-in-itself with respect to the usual topology on $\mathbb{C}$. Then we say that an a-ary function F is pps_ω-definable over $\mathcal{F}$ if there exists a finite subset S_F of $\mathcal{F} \cup \{\mathrm{EQ}\}$ such that, for every $\varepsilon > 0$, there is an a-ary function $\widehat{F}$ specified by a pps-formula over S_F with $\|\widehat{F} - F\|_\infty = \max_{\boldsymbol{x} \in D^a} |\widehat{F}(\boldsymbol{x}) - F(\boldsymbol{x})| < \varepsilon$.

Denote the set of functions in $\mathcal{U}$ that are pps_ω-definable over $\mathcal{F} \cup \{\mathrm{EQ}\}$ by $\langle \mathcal{F} \rangle_\omega$; we call this the pps_ω-*definable functional clone* generated by $\mathcal{F}$. Note that functions in $\langle \mathcal{F} \rangle_\omega$ are determined only by finite subsets of $\mathcal{F}$. Also, although some functions taking values outside R (including partial functions, which are undefined, or infinite, on some inputs) may be pps_ω-definable over $\mathcal{F} \cup \{\mathrm{EQ}\}$, $\langle \mathcal{F} \rangle_\omega$ is defined to include only functions in $\mathcal{U}$. The class of functions $\mathcal{U}$ in operation at any time will be clear from the context.

That completes the setup for expressibility. In order to deduce complexity results, we need an effective version of $\langle \mathcal{F} \rangle_\omega$. We say that a function F is *efficiently* pps_ω-definable over $\mathcal{F}$ if there is a finite subset S_F of $\mathcal{F}$, and a TM $\mathcal{M}_{F,S_F}$ with the following property: on input $\varepsilon > 0$, $\mathcal{M}_{F,S_F}$ computes a pps-formula ψ over S_F such that F_ψ has the same arity as F and $\|F_\psi - F\|_\infty < \varepsilon$. The running time of $\mathcal{M}_{F,S_F}$ is at most a polynomial in $\log \varepsilon^{-1}$. Denote the set of functions in $\mathcal{U}$ that are efficiently pps_ω-definable over $\mathcal{F} \cup \{\mathrm{EQ}\}$ by $\langle \mathcal{F} \rangle_{\omega,p}$; we call this the *efficient* pps_ω-*definable functional clone* generated by $\mathcal{F}$, The following useful observation is immediate from the definition of $\langle \mathcal{F} \rangle_{\omega,p}$.

▶ Observation 2. Suppose $F \in \langle \mathcal{F} \rangle_{\omega,p}$. Then there is a finite $S_F \subseteq \mathcal{F}$ such that $F \in \langle S_F \rangle_{\omega,p}$.

Since pps-formulas are defined using sums of products (with just one level of each), we need to check that functions that are pps_ω-definable in terms of functions that are themselves pps_ω-definable over $\mathcal{F}$ are actually directly pps_ω-definable over $\mathcal{F}$. The following lemma ensures that this is the case.

▶ **Lemma 3.** *If $G \in \langle \mathcal{F} \rangle_\omega$ [or $G \in \langle \mathcal{F} \rangle_{\omega,p}$] then $\langle \mathcal{F}, G \rangle_\omega = \langle \mathcal{F} \rangle_\omega$ [$\langle \mathcal{F}, G \rangle_{\omega,p} = \langle \mathcal{F} \rangle_{\omega,p}$].*

Lemma 3 may have wider applications in the study of approximate counting problems. Often, approximation-preserving reductions between counting problems are complicated

to describe and difficult to analyse, owing to the need to track error estimates. Lemma 3 suggests breaking the reduction into smaller steps, and analysing each of them independently. This assumes, of course, that the reductions are pps_ω-definable, but that often seems to be the case in practice.

3 Relational Clones and Non-negative functions

A function $F \in \mathcal{U}$ is a *Boolean function* if its range is contained in $\{0,1\}$. F encodes a relation R as follows: $\boldsymbol{x}$ is in the relation R iff $F(\boldsymbol{x}) = 1$. We will not distinguish between relations and the Boolean functions that define them. Suppose that $\mathcal{R} \subseteq \mathcal{U}$ is a set of Boolean functions/relations. A pp-formula over $\mathcal{R}$ is an existentially quantified product of atomic formulas. More precisely, a pp-formula ψ over $\mathcal{R}$ in variables $V' = \{v_1, \ldots, v_{n+m}\}$ has the form $\psi = \exists\, v_{n+1}, \ldots, v_{n+m} \bigwedge_{j=1}^{s} \varphi_j$, where φ_j are all atomic formulas over $\mathcal{R}$ in the variables V'. As before, the variables $V = \{v_1, \ldots, v_n\}$ are called "free", and the others, $V' \setminus V$, are called "bound". The formula ψ specifies a Boolean function $R_\psi : D^n \to \{0,1\}$ in the following way. $R_\psi(\boldsymbol{x}) = 1$ if there is a vector $\boldsymbol{y} \in D^m$ such that $\bigwedge_{j=1}^{s} R_{\varphi_j}(\boldsymbol{x}, \boldsymbol{y})$ evaluates to "1", where $\boldsymbol{x}$ and $\boldsymbol{y}$ are assignments $\boldsymbol{x} : \{v_1, \ldots, v_n\} \to D$ and $\boldsymbol{y} : \{v_{n+1}, \ldots, v_{n+m}\} \to D$; $R_\psi(\boldsymbol{x}) = 0$ otherwise. We refer to the pp-formula as an "implementation" of R_ψ.

A *relational clone* (often called a "co-clone") is a set of Boolean relations containing the equality relation and closed under finite Cartesian products, projections, and identification of variables. A *basis* [9] for the relational clone I is a set $\mathcal{R}$ of Boolean relations such that the relations in I are exactly the relations that can be implemented with a pp-formula over $\mathcal{R}$. Every relational clone has such a basis.

For every set $\mathcal{R}$ of Boolean relations, let $\langle \mathcal{R} \rangle_{\mathrm{R}}$ denote the set of relations that can be represented by a pp-formula over $\mathcal{R} \cup \{\mathrm{EQ}\}$. It is well-known that if $R \in \langle \mathcal{R} \rangle_{\mathrm{R}}$ then $\langle \mathcal{R} \cup \{R\} \rangle_{\mathrm{R}} = \langle \mathcal{R} \rangle_{\mathrm{R}}$ Thus, $\langle \mathcal{R} \rangle_{\mathrm{R}}$ is in fact a relational clone with basis $\mathcal{R}$.

A basis $\mathcal{R}$ for a relational clone $\langle \mathcal{R} \rangle_{\mathrm{R}}$ is called a "plain basis" [9, Definition 1] if every member of $\langle \mathcal{R} \rangle_{\mathrm{R}}$ is definable by a CNF($\mathcal{R}$)-formula (a pp-formula over $\mathcal{R}$ with no $\exists$).

For most of this paper, we restrict attention to the Boolean domain $D = \{0,1\}$ and to the codomain $R = \mathbb{R}^{\geq 0}$ of non-negative real numbers. For $n \in \mathbb{N}$, denote by $\mathcal{B}_n$ the set of all functions $\{0,1\}^n \to \mathbb{R}^{\geq 0}$; also denote by $\mathcal{B} = \mathcal{B}_0 \cup \mathcal{B}_1 \cup \mathcal{B}_2 \cup \cdots$ the set of functions of all arities. The advantage of working with the Boolean domain is (i) that it comes with a well-developed theory of relational clones, and (ii) the concept of log-supermodular function makes sense (see §4). As explained in the introduction, the advantage of working with non-negative real numbers is that we thereby forbid cancellation, and potentially obtain a more nuanced expressibility/complexity landscape.

Given a function $F \in \mathcal{B}$, let R_F be the function corresponding to the relation underlying F. That is, $R_F(\boldsymbol{x}) = 0$ if $F(\boldsymbol{x}) = 0$ and $R_F(\boldsymbol{x}) = 1$ if $F(\boldsymbol{x}) > 0$. The following straightforward lemma will be useful.

▶ **Lemma 4.** *Suppose $\mathcal{F} \subseteq \mathcal{B}$. Then $\langle \{R_F \mid F \in \mathcal{F}\} \rangle_{\mathrm{R}} = \{R_F \mid F \in \langle \mathcal{F} \rangle\}$.*

4 Log-supermodular functions

A function $F \in \mathcal{B}_n$ is log-supermodular (lsm) if $F(\boldsymbol{x} \vee \boldsymbol{y})F(\boldsymbol{x} \wedge \boldsymbol{y}) \geq F(\boldsymbol{x})F(\boldsymbol{y})$ for all $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^n$. The terminology is justified by the observation that F is lsm if and only if $f = \ln F$ is supermodular, where $\ln 0$ is treated as $-\infty$, a formal entity that is operated on in the obvious way. We denote by $\mathsf{LSM} \subset \mathcal{B}$ the class of all lsm functions. The second part of our main result (Theorem 8) in some sense says that, in terms of expressivity, everything

of interest takes place in the class LSM. The class LSM fits naturally into our study of expressibility because of the following closure property: functions that are pps_ω-definable from lsm functions are lsm.

▶ **Lemma 5.** *If $\mathcal{F} \subseteq$ LSM is any set of lsm functions then $\langle\mathcal{F}\rangle_\omega \subseteq$ LSM.*

Proof. The only nontrivial step is to show that if $G \in \mathcal{B}_{n+m}$ is lsm then so is the function $G' \in \mathcal{B}_n$ defined by $G'(\boldsymbol{x}) = \sum_{\boldsymbol{y}\in\{0,1\}^m} G(\boldsymbol{x},\boldsymbol{y})$. It is enough to prove the claim for $m = 1$, as the result for general m follows by induction. Suppose $\boldsymbol{a}', \boldsymbol{b}' \in \{0,1\}^n$, and let $A = \{(\boldsymbol{a}', 0), (\boldsymbol{a}', 1)\}$ and $B = \{(\boldsymbol{b}', 0), (\boldsymbol{b}', 1)\}$. We extend G to subsets of $\{0,1\}^{n+1}$ by letting $G(Z) = \sum_{\boldsymbol{z}\in Z} G(\boldsymbol{z})$ for all $Z \subseteq \{0,1\}^{n+1}$. Note that $G'(\boldsymbol{a}') = G(A)$ and $G'(\boldsymbol{b}') = G(B)$. Denote by $A \vee B$ and $A \wedge B$ the sets $A \vee B = \{\boldsymbol{a} \vee \boldsymbol{b} : \boldsymbol{a} \in A \text{ and } \boldsymbol{b} \in B\}$ and $A \wedge B = \{\boldsymbol{a} \wedge \boldsymbol{b} : \boldsymbol{a} \in A \text{ and } \boldsymbol{b} \in B\}$. Note that $G'(\boldsymbol{a}' \vee \boldsymbol{b}') = G(A \vee B)$ and $G'(\boldsymbol{a}' \wedge \boldsymbol{b}') = G(A \wedge B)$. Since G is lsm, we know that $G(\boldsymbol{a})G(\boldsymbol{b}) \le G(\boldsymbol{a} \vee \boldsymbol{b})G(\boldsymbol{a} \wedge \boldsymbol{b})$ for all $\boldsymbol{a}, \boldsymbol{b} \in \{0,1\}^{n+1}$. Thus, applying the Ahlswede-Daykin "Four-functions Theorem" [1, Theorem 1] with $\alpha = \beta = \gamma = \delta = G$,

$$G'(\boldsymbol{a}')G'(\boldsymbol{b}') = G(A)G(B) \le G(A \vee B)G(A \wedge B) = G'(\boldsymbol{a}' \vee \boldsymbol{b}')G'(\boldsymbol{a}' \wedge \boldsymbol{b}').$$

As $\boldsymbol{a}', \boldsymbol{b}' \in \{0,1\}^n$ were arbitrary, G' is lsm. More details are in the full version [4]. ◀

An important example of an lsm function is the 0,1-function "implies", with $\text{IMP}(1,0) = 0$ and $\text{IMP}(x,y) = 1$ for all other x and y. We also think of this as a binary relation $\text{IMP} = \{(0,0), (0,1), (1,1)\}$. Complexity-theoretic issues will be treated in detail in §6. However, it may be helpful to give a pointer here to the importance of IMP in the study of approximate counting problems.

The problem #BIS is that of counting independent sets in a bipartite graph. Dyer et al. [10] exhibited a class of counting problems, including #BIS, which are interreducible via approximation-preserving reductions. Further natural problems have been shown to lie in this class, which appears to be of intermediate complexity between counting problems that are tractable (i.e., admitting a polynomial-time approximation algorithm) and those that are NP-hard to approximate. We will see in due course (Theorem 10) that #BIS and #CSP(IMP) are interreducible via approximation-preserving reductions, and hence are of equivalent difficulty.

We know from Lemma 5 that $\langle\text{IMP}, \mathcal{B}_1\rangle_\omega \subseteq$ LSM. It is an open question whether the inclusion is strict. A related question is whether $\text{LSM} = \langle\mathcal{F}\rangle_\omega$ for any finite set $\mathcal{F}$ of lsm functions. A similar question has been investigated by Živný et al. [19] in this context of optimisation problems, where summation is replaced by maximisation or minimisation.

5 The main result

Since we want to be able to derive computational results, we now restrict attention to functions whose co-domains are restricted to efficiently-computable real numbers. A real number is polynomial-time computable if the first n bits of its binary expansion can be computed in time polynomial in n. Let $\mathbb{R}^p$ denote the set of non-negative real numbers that are polynomial-time computable. For $n \in \mathbb{N}$, denote by $\mathcal{B}^p_n$ the set of all functions $\{0,1\}^n \to \mathbb{R}^p$; also denote by $\mathcal{B}^p = \mathcal{B}^p_0 \cup \mathcal{B}^p_1 \cup \mathcal{B}^p_2 \cup \cdots$ the set of functions of all arities.

▶ Remark. If $\mathcal{F} \subseteq \mathcal{B}^p$ then real numbers appearing as function values must be polynomial-time computable. This is a stronger requirement than the *efficiently approximable* real numbers defined in [13], but it results in a more uniform treatment of limits when we discuss efficient pps_ω-definability using these functions.

5.1 Pinnings and modular functions

Let δ_0 be the unary function with $\delta_0(0) = 1$ and $\delta_0(1) = 0$ and let δ_1 be the unary function with $\delta_1(0) = 0$ and $\delta_1(1) = 1$.

If $n \geq 2$ then a 2-*pinning* of a function $F \in \mathcal{B}_n$ is a function

$$G_{i,j}(x_1, x_2) = F(c_1, \ldots, c_{i-1}, x_1, c_{i+1}, \ldots, c_{j-1}, x_2, c_j, \ldots, c_n),$$

where $i, j \in \{1, \ldots, n\}$, $i \neq j$, and each c_k is in $\{0, 1\}$. Clearly, every 2-pinning of F is in $\langle F, \mathcal{B}_1^p \rangle$, since the constants c_k can be implemented using the functions δ_0 and δ_1.

We say that a function $F \in \mathcal{B}_n$ is log-modular if $f = \ln F$ is modular, ie., $F(\boldsymbol{x} \vee \boldsymbol{y})F(\boldsymbol{x} \wedge \boldsymbol{y}) = F(\boldsymbol{x})F(\boldsymbol{y})$ for all $\boldsymbol{x}, \boldsymbol{y} \in \{0, 1\}^n$.

We will use the following fact about 2-pinnings of lsm and log-modular functions. This follows directly from [15, Theorem 44.1] for the supermodular case, but Schrijver's proof also applies to the modular case. The lemma is originally due to Topkis [16].

▶ **Lemma 6** (Topkis). *Let F be a function from $\{0, 1\}^n$ to $\mathbb{R}^{>0}$. F is lsm iff every 2-pinning of F is lsm. F is log-modular iff every 2-pinning of F is log-modular.*

5.2 Binary functions

Recall that EQ is the binary relation $\mathrm{EQ} = \{(0, 0), (1, 1)\}$. (We used the name "EQ" to denote the equivalent binary function as well.) Denote by OR, NEQ, and NAND the binary relations $\mathrm{OR} = \{(0, 1), (1, 0), (1, 1)\}$, $\mathrm{NEQ} = \{(0, 1), (1, 0)\}$, and $\mathrm{NAND} = \{(0, 0), (0, 1), (1, 0)\}$.

▶ **Lemma 7.** *Let $F \in \mathcal{B}_2^p$. Assuming $F(0, 1) \geq F(1, 0)$,*

(i) *if $F(0, 0)F(1, 1) = F(0, 1)F(1, 0)$, then $\langle F, \mathcal{B}_1^p \rangle_{\omega,p} = \langle \mathcal{B}_1^p \rangle_{\omega,p}$;*
(ii) *if $R_F = \mathrm{EQ}$, then $\langle F, \mathcal{B}_1^p \rangle_{\omega,p} = \langle \mathcal{B}_1^p \rangle_{\omega,p}$;*
(iii) *if $R_F = \mathrm{NEQ}$, then $\langle F, \mathcal{B}_1^p \rangle_{\omega,p} = \langle \mathrm{NEQ}, \mathcal{B}_1^p \rangle_{\omega,p}$;*
(iv) *if $\mathrm{IMP} \subseteq R_F$ and $F(0, 0)F(1, 1) > F(0, 1)F(1, 0)$, then $\langle F, \mathcal{B}_1^p \rangle_{\omega,p} = \langle \mathrm{IMP}, \mathcal{B}_1^p \rangle_{\omega,p}$;*
(v) *otherwise, $\langle F, \mathcal{B}_1^p \rangle_{\omega,p} = \langle \mathrm{OR}, \mathcal{B}_1^p \rangle_{\omega,p} = \mathcal{B}^p$.*

▶ Remark. From Lemma 7, we see that IMP does not really occupy a special position in $\langle \mathrm{IMP}, \mathcal{B}_1^p \rangle_{\omega,p}$, in the sense that there are other function F with $\langle F, \mathcal{B}_1^p \rangle_{\omega,p} = \langle \mathrm{IMP}, \mathcal{B}_1^p \rangle_{\omega,p}$. Similarly, OR does not occupy a special position in $\langle \mathrm{OR}, \mathcal{B}_1^p \rangle_{\omega,p}$. Nevertheless, it is useful to label the classes this way, and we will do so.

▶ Remark. From the proof of Lemma 7, we have the following inclusions between the four classes involved. $\langle \mathcal{B}_1^p \rangle_{\omega,p} \subseteq \langle \mathrm{NEQ}, \mathcal{B}_1^p \rangle_{\omega,p} \subseteq \langle \mathrm{OR}, \mathcal{B}_1^p \rangle_{\omega,p}$ and $\langle \mathcal{B}_1^p \rangle_{\omega,p} \subseteq \langle \mathrm{IMP}, \mathcal{B}_1^p \rangle_{\omega,p} \subseteq \langle \mathrm{OR}, \mathcal{B}_1^p \rangle_{\omega,p}$. In fact, $\langle \mathrm{NEQ}, \mathcal{B}_1^p \rangle_{\omega,p}$ and $\langle \mathrm{IMP}, \mathcal{B}_1^p \rangle_{\omega,p}$ are incomparable, and hence all the inclusions are actually strict. For one non-inclusion, note the clone $\langle \mathrm{IMP}, \mathcal{B}_1^p \rangle_{\omega,p}$ contains only lsm functions, and hence does not contain NEQ. For the other, we claim that arity-2 functions in the clone $\langle \mathrm{NEQ}, \mathcal{B}_1^p \rangle_{\omega,p}$ are of one of three forms — $U_1(x)U_2(y)$, $U(x)\mathrm{EQ}(x, y)$ or $U(x)\mathrm{NEQ}(x, y)$ — and then observe that IMP matches none of these.

5.3 Functional clones on 2-element set

▶ **Theorem 8.** *Suppose $F \in \mathcal{B}^p$.*

- *If $F \notin \langle \mathrm{NEQ}, \mathcal{B}_1^p \rangle$ then $\mathrm{IMP} \in \langle F, \mathcal{B}_1^p \rangle_{\omega,p}$, and hence $\langle \mathrm{IMP}, \mathcal{B}_1^p \rangle_{\omega,p} \subseteq \langle F, \mathcal{B}_1^p \rangle_{\omega,p}$*
- *If, in addition, $F \notin \mathsf{LSM}$ then $\langle F, \mathcal{B}_1^p \rangle_{\omega,p} = \mathcal{B}^p$.*

The non-effective version of the theorem — with $\mathcal{B}, \mathcal{B}_1$ replacing $\mathcal{B}^p, \mathcal{B}_1^p$, and $\langle \cdot \rangle_\omega$ replacing $\langle \cdot \rangle_{\omega,p}$ — also holds.

Proof. We start with the first part of the theorem, for which the aim is to show that either IMP $\in \langle F, \mathcal{B}_1^p\rangle_{\omega,p}$ or $F \in \langle \mathrm{NEQ}, \mathcal{B}_1^p\rangle$. Let C be the relational clone $\langle R_F, \delta_0, \delta_1\rangle_{\mathrm{R}}$. Since $\{R_F, \delta_0, \delta_1\} \subseteq \{R_{F'} \mid F' \in \{F\} \cup \mathcal{B}_1^p\}$, $C \subseteq \langle R_{F'} \mid F' \in \{F\} \cup \mathcal{B}_1^p\rangle_{\mathrm{R}}$, so by Lemma 4, $C \subseteq \{R_{F'} \mid F' \in \langle F, \mathcal{B}_1^p\rangle\}$.

First, suppose IMP $\in C$. Then $\langle F, \mathcal{B}_1^p\rangle_{\omega,p}$ contains a function F' with $R_{F'} = \mathrm{IMP}$. The function F' falls into parts (iv) or (v) of Lemma 7, so by this lemma, $\langle F, \mathcal{B}_1^p\rangle_{\omega,p}$ is either $\langle \mathrm{IMP}, \mathcal{B}_1^p\rangle_{\omega,p}$ or $\langle \mathrm{OR}, \mathcal{B}_1^p\rangle_{\omega,p}$. Either way, $\langle F, \mathcal{B}_1^p\rangle_{\omega,p}$ contains IMP (as noted in Remark at the end of Section 5.2). Similarly, if OR $\in C$ or NAND $\in C$ then IMP $\in \langle F, \mathcal{B}_1^p\rangle_{\omega,p}$.

We now consider the possibilities. If R_F is not affine, then [8, Lemma 5.30] shows that one of IMP, OR and NAND is in C. This is also proved in [11, Lemma 15].

In fact, the set of all relational clones (also called "co-clones") is well understood. These are listed in [9, Table 2], which gives a plain basis for each relational clone. A graph illustrating the subset inclusions between the relational clones, called Post's lattice, is depicted in [2, Figure 2]. This graph is reproduced in the full version [4, Figure 1]. The relational clones are the vertices of the graph. A downwards edge from one clone to another indicates that the lower clone is a subset of the higher one. For example, since there is a path (in this case, an edge) from ID_1 down to IR_2 in Post's lattice, we deduce that $\mathrm{IR}_2 \subset \mathrm{ID}_1$. We will require bases for only 3 relational clones: IR_2, ID_1, and IL_2; their plain bases are $\{\mathrm{EQ}, \delta_0, \delta_1\}$, $\{\mathrm{EQ}, \mathrm{NEQ}, \delta_0, \delta_1\}$, and $\{(x_1 \oplus \ldots \oplus x_k = c) \mid k \in \mathbb{N}, c \in \{0,1\}\}$, respectively.

If R_F is affine then the relations in C are given by linear equations, so C is either the relational clone IL_2 (whose plain basis is the set of all Boolean linear equations) or C is some subset of IL_2, in which case it is below IL_2 in [4, Figure 1].

Now, EQ, δ_0 and δ_1 are in C. The relational clone containing these relations (and nothing else) is IR_2, so C is a (not necessarily proper) superset of IR_2. Thus, C is (not necessarily strictly) above IR_2 in [4, Figure 1]. From the figure, it is clear that the only possibilities are that C is one of the relational clones IL_2, ID_1 and IR_2.

Now $\mathrm{IR}_2 \subset \mathrm{ID}_1$ and the plain basis of ID_1 is $\{\mathrm{EQ}, \mathrm{NEQ}, \delta_0, \delta_1\}$. Therefore if $C = \mathrm{IR}_2$ or $C = \mathrm{ID}_1$, then R_F is definable by a CNF formula over $\{\mathrm{EQ}, \mathrm{NEQ}, \delta_0, \delta_1\}$.

Suppose that $F(\boldsymbol{x})$ has arity n. To avoid trivialities, suppose that R_F is not the empty relation. Suppose that $\psi(v_1, \ldots, v_n)$ is a CNF formula over $\{\mathrm{EQ}, \mathrm{NEQ}, \delta_0, \delta_1\}$ implementing the relation $R_\psi = R_F$.

Let $V = \{v_1, \ldots, v_n\}$. Let ψ_i be the projection of ψ onto variable v_i. ψ_i is one of the three unary relations $\{(0)\}$, $\{(1)\}$, and $\{(0),(1)\}$. Let $V' = \{v_i \in V \mid \psi_i = \{(0),(1)\}\}$. For $v_i \in V'$ and $v_j \in V'$, let $\psi_{i,j}$ be the projection of ψ onto variables v_i and v_j. $\psi_{i,j}$ is a binary relation. As is easily seen, of the 16 possible binary relations, the only ones that can occur are EQ, NEQ and $\{0,1\}^2$. (See the full version [4] for details.) We define an equivalence relation $\sim$ on V' in which $v_i \sim v_j$ iff $\psi_{i,j} \in \{\mathrm{EQ}, \mathrm{NEQ}\}$. Let V'' contain exactly one variable from each equivalence class in V'. Let $k = |V''|$. For convenience, we will assume $V'' = \{v_1, \ldots, v_k\}$.

Now, for every assignment $\boldsymbol{x} : \{v_1, \ldots, v_k\} \to \{0,1\}$ there is exactly one assignment $\boldsymbol{y} : \{v_{k+1}, \ldots, v_n\} \to \{0,1\}$ such that $R_F(\boldsymbol{x}, \boldsymbol{y}) = 1$. Let $\sigma(\boldsymbol{x})$ be this assignment $\boldsymbol{y}$. Now, define the arity-k function G by $G(\boldsymbol{x}) = F(\boldsymbol{x}, \sigma(\boldsymbol{x}))$. Note that

$$G(\boldsymbol{x}) = \sum_{\boldsymbol{y} \in \{0,1\}^{n-k}} F(\boldsymbol{x}, \boldsymbol{y}), \tag{2}$$

where $\boldsymbol{y}$ is an assignment $\boldsymbol{y} : \{v_{k+1}, \ldots, v_n\} \to \{0,1\}$. By construction, $G(\boldsymbol{x})$ is a strictly positive function. Also, from (2), $G \in \langle F, \mathcal{B}_1^p\rangle_{\omega,p}$. We finish with two cases.

Case 1. Every 2-pinning of G is log-modular. Then G is also log-modular, by Lemma 6. This means (see, for example, [3, Proposition 24]) that $g = \ln G$ is a linear function

of $x_1, \ldots, x_k$ and $\neg x_1, \ldots, \neg x_k$ so $G \in \langle \mathrm{NEQ}, \mathcal{B}_1^p \rangle$. (For example, if $g = a_1 x_1 + a_2 x_2 + a_3 \neg x_3$ then G can be written as $G(x_1, x_2, x_3) = \sum_{y_3} f_1(x_1) f_2(x_2) f_3(y_3) \mathrm{NEQ}(x_3, y_3)$, where $f_i(x) = \exp(a_i x)$ is a function in $\mathcal{B}_1^p$.) Since $F(\boldsymbol{x}, \boldsymbol{y}) = R_F(\boldsymbol{x}, \boldsymbol{y}) G(\boldsymbol{x})$, we conclude that $F \in \langle \mathrm{NEQ}, \mathcal{B}_1^p \rangle$.

Case 2. There is a 2-pinning G' of G that is not log-modular. Since G is strictly positive, so is G'. Since $G \in \langle F, \mathcal{B}_1^p \rangle_{\omega,p}$, so is G'. By Lemma 7, (parts (iv) or (v)), $\mathrm{IMP} \in \langle G', \mathcal{B}_1^p \rangle_{\omega,p}$. By Lemma 3, $\mathrm{IMP} \in \langle F, \mathcal{B}_1^p \rangle_{\omega,p}$.

Finally, we consider the case in which $C = \mathrm{IL}_2$. Let $\oplus_3$ be the relation $\{(0,0,0), (0,1,1), (1,0,1), (1,1,0)\}$ containing all triples whose Boolean sums are 0. From the plain basis of IL_2, we see that the relation $\oplus_3$ is in C, so $\langle F, \mathcal{B}_1^p \rangle$ contains a function F' with $R_{F'} = \oplus_3$. Let F'' be the symmetrisation of F' implemented by

$$F''(x,y,z) = F'(x,y,z)F'(x,z,y)F'(y,x,z)F'(y,z,x)F'(z,x,y)F'(z,y,z).$$

Now let $\mu_0 = F''(0,0,0)$ and $\mu_2 = F''(0,1,1)$. Let U be the unary function with $U(0) = \mu_0^{-1/3}$ and $U(1) = \mu_0^{1/6} \mu_2^{-1/2}$. Note that since $F \in \mathcal{B}^p$, the appropriate roots of μ_0 and μ_2 are efficiently computable, so $U \in \mathcal{B}_1^p$. Now $\oplus_3(x,y,z) = U(x)U(y)U(z)F''(x,y,z)$, so $\oplus_3 \in \langle F, \mathcal{B}_1^p \rangle$. Finally, let U' be the unary function defined by $U'(0) = 1$ and $U'(1) = 2$ and let $G(x,z) = \sum_y \oplus_3(x,y,z)U'(y)$. Note that $G(0,0) = G(1,1) = 1$ and $G(0,1) = G(1,0) = 2$. By Lemma 1, G is in $\langle F, \mathcal{B}_1^p \rangle$. But by Lemma 7, $\mathrm{IMP} \in \langle G, \mathcal{B}_1^p \rangle_{\omega,p}$ so by Lemma 3, $\mathrm{IMP} \in \langle F, \mathcal{B}_1^p \rangle_{\omega,p}$.

We now prove Part 2 of the theorem. Suppose that F is not lsm and that $F \notin \langle \mathrm{NEQ}, \mathcal{B}_1^p \rangle$ so, by Part 1 of the theorem, we have $\mathrm{IMP} \in \langle F, \mathcal{B}_1^p \rangle_{\omega,p}$. Let $H(x_1, x_2) = \sum_{y_1, y_2} \mathrm{IMP}(y_1, x_1)\mathrm{IMP}(y_1, x_2)\mathrm{IMP}(x_1, y_2)\mathrm{IMP}(x_2, y_2)$. Note that $H(0,0) = H(1,1) = 2$ and $H(0,1) = H(1,0) = 1$. Now for any integer k, let

$$H_k(x_1, \ldots, x_n) = \sum_{y_1, \ldots, y_n} F(y_1, \ldots, y_n) \prod_{i=1}^{n} H(x_i, y_i)^k.$$

By construction, H_k is strictly positive. Also, as k gets large, $H_k(x_1, \ldots, x_n)$ gets closer and closer to $2^{kn} F(x_1, \ldots, x_n)$. Thus, for sufficiently large k, H_k is not lsm. By Lemma 1, $H \in \langle F, \mathcal{B}_1^p \rangle_{\omega,p}$ so $H_k \in \langle F, \mathcal{B}_1^p \rangle_{\omega,p}$. Applying Lemma 6 to H_k, there is a binary function $F_1 \in \langle F, \mathcal{B}_1^p \rangle$ that is not lsm so $F_1(0,0)F_1(1,1) < F_1(0,1)F_1(1,0)$. By Parts (iii) and (v) of Lemma 7, we either have $\mathrm{NEQ} \in \langle F, \mathcal{B}_1^p \rangle$ or $\mathrm{OR} \in \langle F, \mathcal{B}_1^p \rangle$. In the latter case, we are finished by (v) of Lemma 7. In the former case, we are also finished since $\mathrm{OR} \in \langle \mathrm{IMP}, \mathrm{NEQ} \rangle$— details are given in the full version [4]. ◀

6 Complexity-theoretic consequences

In order to explore the computational consequences of Theorem 8, we need to recall some definitions from computational complexity, specifically relating to approximate counting problems. For contextual material and proofs of any unsubstantiated claims made below, please refer to [10].

For our purposes, a counting problem is a function Π from instances w (encoded as a word over some alphabet Σ) to a number $\Pi(w) \in \mathbb{R}^{\geq 0}$. For example, w might encode an instance I of a counting CSP problem $\#\mathsf{CSP}(\Gamma)$, in which case $\Pi(w)$ would be the partition function $Z(I)$ associated with I. A *randomised approximation scheme* (RAS) for Π is a randomised algorithm that takes an instance w and returns an approximation Y to $\Pi(w)$. The approximation scheme has a parameter $\varepsilon > 0$ which specifies the error tolerance. Since

the algorithm is randomised, the output Y is a random variable depending on the "coin tosses" made by the algorithm. We require that, for every instance w and every $\varepsilon > 0$, $\Pr\left[e^{-\varepsilon}\Pi(w) \leq Y \leq e^{\varepsilon}\Pi(w)\right] \geq 3/4$. The RAS is said to be a *fully polynomial randomised approximation scheme*, or *FPRAS*, if it runs in time bounded by a polynomial in $|w|$ (the length of the word w) and ε^{-1}. See Mitzenmacher and Upfal [14, Definition 10.2].

Suppose that Π_1 and Π_2 are functions from Σ^* to $\mathbb{R}^{\geq 0}$. An "approximation-preserving reduction" (AP-reduction) [10] from Π_1 to Π_2 gives a way to turn an FPRAS for Π_2 into an FPRAS for Π_1. Specifically, an *AP-reduction from Π_1 to Π_2* is a randomised algorithm $\mathcal{A}$ for computing Π_1 using an oracle[1] for Π_2. The algorithm $\mathcal{A}$ takes as input a pair $(w, \varepsilon) \in \Sigma^* \times (0, 1)$, and satisfies the following three conditions: (i) every oracle call made by $\mathcal{A}$ is of the form (v, δ), where $v \in \Sigma^*$ is an instance of Π_2, and $0 < \delta < 1$ is an error bound satisfying $\delta^{-1} \leq \text{poly}(|w|, \varepsilon^{-1})$; (ii) the algorithm $\mathcal{A}$ meets the specification for being a randomised approximation scheme for Π_1 (as described above) whenever the oracle meets the specification for being a randomised approximation scheme for Π_2; and (iii) the run-time of $\mathcal{A}$ is polynomial in $|w|$ and ε^{-1}. Note that the class of functions computable by an FPRAS is closed under AP-reducibility. Informally, AP-reducibility is the most liberal notion of reduction meeting this requirement. If an AP-reduction from Π_1 to Π_2 exists we write $\Pi_1 \leq_{\text{AP}} \Pi_2$. If $\Pi_1 \leq_{\text{AP}} \Pi_2$ and $\Pi_2 \leq_{\text{AP}} \Pi_1$ then we say that Π_1 *and* Π_2 *are AP-interreducible*, and write $\Pi_1 =_{\text{AP}} \Pi_2$.

A word of warning about terminology. Subsequent to [10] the notation $\leq_{\text{AP}}$ has been used to denote a different type of approximation-preserving reduction which applies to optimisation problems. We will not study optimisation problems in this paper, so hopefully this will not cause confusion.

The complexity of approximating Boolean #CSPs in the unweighted case (i.e., where the functions in Γ have codomain $\{0, 1\}$) was earlier studied by the final three authors [11]. Two counting problems played a special role there, and in earlier work in the complexity of approximate counting [10]. They also play a key role here.

Name #SAT
Instance A Boolean formula φ in conjunctive normal form.
Output The number of satisfying assignments of φ.

Name #BIS
Instance A bipartite graph B.
Output The number of independent sets in B.

An FPRAS for #SAT would, in particular, have to decide with high probability between a formula having some satisfying assignments or having none. Thus #SAT cannot have an FPRAS unless NP = RP.[2] The same is true of any problem to which #SAT is AP-reducible. As far as we are aware, the complexity of approximating #BIS does not relate to any of the standard complexity theoretic assumptions, such as NP ≠ RP. Nevertheless, there is increasing empirical evidence that no FPRAS for #BIS exists, and we adopt this as a working hypothesis. Of course, this hypothesis implies that no #BIS-hard problem (problem to which #BIS is AP-reducible) admits an FPRAS. Finally, here is a precise statement of the

[1] The reader who is not familiar with oracle Turing machines can just think of this as an imaginary (unwritten) subroutine for computing Π_2.
[2] The supposed FPRAS would provide a polynomial-time decision procedure for satisfiability with two-sided error; however, there is a standard trick for converting two-sided error to the one-sided error demanded by the definition of RP [17, Thm 10.5.9].

computational task we are interested in. A (weighted) #CSP problem is parameterised by a finite subset $\mathcal{F}$ of $\mathcal{B}^p$ and defined as follows.

Name $\#\mathsf{CSP}(\mathcal{F})$
Instance A pps-formula ψ consisting of a product of m atomic $\mathcal{F}$-formulas over n free variables $\boldsymbol{x}$. (Thus, ψ has no bound variables.)
Output The value $\sum_{\boldsymbol{x}\in\{0,1\}^n} F_\psi(\boldsymbol{x})$ where F_ψ is the function defined by that formula.

Officially, the input size $|w|$ is the length of the encoding of the instance. However, we shall take the size of a $\#\mathsf{CSP}(\mathcal{F})$ instance to be $n+m$, where n is the number of (free) variables and m is the number of constraints (atomic formulas). This is acceptable, as we are only concerned to measure the input size within a polynomial factor; moreover, we have restricted Γ to be finite, thereby avoiding the issue of how to the encode constraint functions $\mathcal{F}$. We typically denote an instance of $\#\mathsf{CSP}(\mathcal{F})$ by I and the output by $Z(I)$; by analogy with systems in statistical physics we refer to $Z(I)$ as the partition function.

Aside from simplifying the representation of problem instances, there is another, more important reason for decreeing that $\mathcal{F}$ is finite, namely, that it allows us to prove the following basic lemma relating functional clones and computational complexity. It is, of course, based on a similar result for classical decision CSPs.

▶ **Lemma 9.** *Suppose $\mathcal{F}\subseteq\mathcal{B}^p$ is finite. If $F\in\langle\mathcal{F}\rangle_{\omega,p}$ then $\#\mathsf{CSP}(F,\mathcal{F})\leq_{\mathrm{AP}}\#\mathsf{CSP}(\mathcal{F})$*

Proof. Let k be the arity of F. Let $\mathcal{M}$ be a TM which, on input $\varepsilon'>0$, computes a k-ary pps-formula ψ over $\mathcal{F}\cup\mathrm{EQ}$ such that $\|F_\psi - F\|_\infty<\varepsilon'$. Consider an input (I,ε) where I is an instance of $\#\mathsf{CSP}(F,\mathcal{F})$ and ε is an accuracy parameter. The key idea of the proof is to construct an instance I' of $\#\mathsf{CSP}(\mathcal{F})$ by replacing each F-constraint in I with the set of constraints and extra (bound) variables in the formula ψ that is output by $\mathcal{M}$ with input ε'. After choosing an appropriate ε' the proof can be completed by a fairly straightforward computation (see the full version [4]). ◀

▶ **Theorem 10.** *Suppose $\mathcal{F}$ is a finite subset of $\mathcal{B}^p$.*

- *If $\mathcal{F}\subseteq\langle\mathrm{NEQ},\mathcal{B}_1^p\rangle$ then, for any finite $S\subseteq\mathcal{B}_1^p$, there is an FPRAS for $\#\mathsf{CSP}(\mathcal{F},S)$.*
- *Otherwise,*
  - *There is a finite subset S of $\mathcal{B}_1^p$ such that $\#\mathsf{BIS}\leq_{\mathrm{AP}}\#\mathsf{CSP}(\mathcal{F},S)$.*
  - *If there is a function $F\in\mathcal{F}$ such that $F\notin\mathsf{LSM}$ then there is a finite subset S of $\mathcal{B}_1^p$ such that $\#\mathsf{SAT}=_{\mathrm{AP}}\#\mathsf{CSP}(\mathcal{F},S)$.*

▶ **Example 11.** Let $F\in\mathcal{B}_2^p$ be the function defined by $F(0,0)=F(1,1)=\lambda$ and $F(0,1)=F(1,0)=1$, where $\lambda>1$. Then, by Theorem 10, $\#\mathsf{CSP}(F,S)$ is $\#\mathsf{BIS}$-hard, for some set S of unary weights. (This counting CSP is also $\#\mathsf{BIS}$-easy.) Note that $\#\mathsf{CSP}(F,S)$ is nothing other than the ferromagnetic Ising model with an applied field. So we recover, with no effort, the main result of Goldberg and Jerrum's investigation of this model [12].

▶ **Example 12.** If F is as before, but $\lambda\in(0,1)$, then $F\notin\mathsf{LSM}$ and Theorem 10 tells us that $\#\mathsf{CSP}(F,S)$ is $\#\mathsf{SAT}$-hard, for some set S of unary weights. This is a restatement of the well-known fact that the partition function of an antiferromagnetic Ising model is hard to compute, even approximately.

References

1 Rudolf Ahlswede and David E. Daykin. An inequality for the weights of two families of sets, their unions and intersections. *Z. Wahrsch. Verw. Gebiete*, 43(3):183–185, 1978.

2 Elmar Böhler, Nadia Creignou, Steffen Reith, and Heribert Vollmer. Playing with Boolean blocks, part II: Constraint satisfaction problems. *ACM SIGACT-Newsletter*, 35:22–35, 2004.

3 Endre Boros and Peter L. Hammer. Pseudo-boolean optimization. *Discr. Appl. Math.*, 123(1-3):155–225, 2002.

4 A. A. Bulatov, M. Dyer, L. A. Goldberg, and M. Jerrum. Log-supermodular functions, functional clones and counting CSPs. *ArXiv e-prints*, August 2011.

5 Jin-Yi Cai, Xi Chen, and Pinyan Lu. Non-negative weighted #CSPs: An effective complexity dichotomy. *CoRR*, abs/1012.5659, 2010.

6 Jin-Yi Cai, Pinyan Lu, and Xia Mingji. Dichotomy for Holant* problems of Boolean domain. In *SODA*, pages 1714–1728, 2011.

7 David Cohen and Peter Jeavons. Chapter 8: The complexity of constraint languages. In Peter van Beek Francesca Rossi and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 245 – 280. Elsevier, 2006.

8 Nadia Creignou, Sanjeev Khanna, and Madhu Sudan. *Complexity Classifications of Boolean Constraint Satisfaction Problems*. SIAM, Philadelphia, PA, USA, 2001.

9 Nadia Creignou, Phokion Kolaitis, and Bruno Zanuttini. Structure identification of Boolean relations and plain bases for co-clones. *Journal of Computer and System Sciences*, 74(7):1103 – 1115, 2008.

10 Martin Dyer, Leslie Ann Goldberg, Catherine Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004. Approximation algorithms.

11 Martin Dyer, Leslie Ann Goldberg, and Mark Jerrum. An approximation trichotomy for Boolean #CSP. *Journal of Computer and System Sciences*, 76(3-4):267 – 277, 2010.

12 Leslie Ann Goldberg and Mark Jerrum. The complexity of ferromagnetic Ising with local fields. *Combin. Probab. Comput.*, 16(1):43–61, 2007.

13 Leslie Ann Goldberg and Mark Jerrum. Approximating the partition function of the ferromagnetic Potts model. In Samson Abramsky, Cyril Gavoille, Claude Kirchner, Friedhelm Meyer auf der Heide, and Paul G. Spirakis, editors, *ICALP (1)*, volume 6198 of *LNCS*, pages 396–407. Springer, 2010.

14 Michael Mitzenmacher and Eli Upfal. *Probability and Computing*. Cambridge University Press, Cambridge, 2005. Randomized algorithms and probabilistic analysis.

15 Alexander Schrijver. *Combinatorial Optimization - Polyhedra and Efficiency*. Springer, 2003.

16 Donald M. Topkis. Minimizing a submodular function on a lattice. *Operat. Res.*, 26:305–321, 1978.

17 Ingo Wegener. *Complexity Theory*. Springer-Verlag, Berlin, 2005. Exploring the limits of efficient algorithms, Translated from the German by Randall Pruim.

18 Tomoyuki Yamakami. Approximate counting for complex-weighted Boolean constraint satisfaction problems. *CoRR*, abs/1007.0391, 2010.

19 Stanislav Živný, David A. Cohen, and Peter G. Jeavons. The expressive power of binary submodular functions. *Discrete Appl. Math.*, 157(15):3347–3358, 2009.

Low Randomness Rumor Spreading via Hashing

George Giakkoupis*[1], Thomas Sauerwald[2], He Sun[2,3], and Philipp Woelfel†[1]

1 **Computer Science Department, University of Calgary, Canada**
{ggiakkou, woelfel}@ucalgary.ca
2 **Max Planck Institute for Informatics, Germany**
{sauerwal, hsun}@mpi-inf.mpg.de
3 **Institute of Modern Mathematics and Physics, Fudan University, China**

Abstract

We consider the classical rumor spreading problem, where a piece of information must be disseminated from a single node to all n nodes of a given network. We devise two simple push-based protocols, in which nodes choose the neighbor they send the information to in each round using pairwise independent hash functions, or a pseudo-random generator, respectively. For several well-studied topologies our algorithms use exponentially fewer random bits than previous protocols. For example, in complete graphs, expanders, and random graphs only a polylogarithmic number of random bits are needed in total to spread the rumor in $\mathcal{O}(\log n)$ rounds with high probability. Previous explicit algorithms, e.g., [10, 17, 6, 15], require $\Omega(n)$ random bits to achieve the same round complexity. For complete graphs, the amount of randomness used by our hashing-based algorithm is within an $\mathcal{O}(\log n)$-factor of the theoretical minimum determined by Giakkoupis and Woelfel [15].

1998 ACM Subject Classification G.3 Probability and Statistics

Keywords and phrases Parallel and Distributed Computing, Randomness, Rumor Spreading

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.314

1 Introduction

Broadcasting a piece of information to all nodes in a network is one of the fundamental problems in the theory of network algorithms. A basic variant of the problem is the *rumor spreading problem*: One node in a graph with n nodes initially obtains a piece of information, called the *rumor*. In subsequent synchronous rounds, nodes communicate with randomly chosen neighbors in order to spread the rumor. Protocols for rumor spreading are of fundamental interest and have several applications, such as in the maintenance of distributed replicated database systems [4, 10], failure detection [24], resource discovery [16], and data aggregation [1]. As such, the problem has been well-studied in the literature.

Several design goals have been considered when devising rumor spreading protocols: Most importantly, the algorithm should be *efficient*, in the sense that the rumor reaches every node in a small number of rounds. In addition, rumor spreading protocols should be *local*, i.e., nodes should not need to have any information about the global connectivity of the network. Another important property is *robustness*, that is, the protocol can tolerate the failure of some links [10, 17].

* Supported by the Pacific Institute for the Mathematical Sciences (PIMS), and the Natural Sciences and Engineering Research Council of Canada (NSERC).
† Supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 314–325

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

Under standard model assumptions, *deterministic* rumor spreading protocols are often inefficient. For example, on complete graphs, it is not possible to spread the rumor in $o(n)$ rounds if nodes in the adjacency lists can be ordered arbitrarily (and nodes do not know the orderings). Hence, essentially all rumor spreading protocols of note are *randomized*.

In the standard models, a node u can open a communication channel to one of its neighbors, v. If u knows the rumor, u can then *push* it to v, and if u does not know the rumor, it can try to *pull* it from v. In *push-protocols*, where the rumor is disseminated solely by push communication, it is easy to share randomness among all nodes: The first node that obtains the rumor can generate a random string and then nodes pass the same random string along with the rumor. It is known that for push-protocols it is necessary to share randomness in order to achieve both, time and randomness efficiency [15]. On the other hand, in order to benefit from pull-communications, nodes that have not received any messages would have to generate their own "private" random strings to determine their random communication partners. Thus, it seems that pull-communications cannot help to spread the rumor to many nodes, unless many nodes generate random strings, and thus in total a large number of random bits is generated. Therefore, in this paper we restrict our attention to push-protocols.

Precisely, a push-protocol proceeds in synchronous rounds as follows: Initially, in round 0, an arbitrary node receives the rumor. In every succeeding round, every *informed* node (i.e., every node that received the rumor in a previous round) chooses a random neighbor (according to some probability distribution), which it then informs about the rumor.

In the *fully random* protocol, in each round every informed node chooses its neighbor uniformly at random. This simple classical protocol, which has been extensively studied, is local, robust, and efficient. For instance, for a variety of graphs, such as complete graphs [13, 23], hypercubes [10], random graphs [12], expanders [21, 14], or regular graphs with constant conductance [14], only $\mathcal{O}(\log n)$ rounds are needed to spread the rumor to all nodes with high probability (w.h.p.).[1] In a graph of degree d, every informed node needs to choose $\log d$ random bits in every round, and thus typically a total of $\Theta(t \cdot n \cdot \log d)$ random bits are needed, if the protocol runs for t rounds.

The so-called *quasi-random* protocol was proposed by Doerr, Friedrich, and Sauerwald with the aim of "imitating properties of the classical push model with a much smaller degree of randomness" [6]. The idea is that each node, once it becomes informed, only chooses one starting point in its adjacency list uniformly at random. From then on, it contacts its neighbors in the order they appear in the adjacency list, beginning with that starting point (and in a round-robin fashion). The protocol has very similar properties as the fully random algorithm, and in particular it has been proven to be as efficient on complete graphs, random graphs, strong expanders and hypercubes (see also Table 1). Only $\mathcal{O}(n \log d)$ random bits are needed in total for the quasi-random protocol on a graph of degree d.

Doerr and Fouz [5] showed that in the complete graph one cannot further reduce the amount of randomness of the quasi-random protocol by limiting each node's choice of its starting point in its list without sacrificing the efficiency of the protocol. Giakkoupis and Woelfel [15] proved more general upper and lower bounds for the amount of randomness required to spread a rumor on the complete graph: They provided a relatively simple protocol that needs only $\mathcal{O}(n \log \log n)$ random bits in total to spread the rumor to all nodes of the complete graph. Moreover, they showed that any protocol that uses only $\log n - \log \log n - \omega(1)$ random bits needs $\omega(\log n)$ rounds to inform all nodes. While the probabilistic method can

[1] We say an event occurs with high probability, if there exists a constant $\varepsilon > 0$ such that the probability of the event is $1 - \mathcal{O}(n^{-\varepsilon})$.

Graph family	Rumor spreading time		Random bits	Reference
Graphs with $\Delta/\delta = \mathcal{O}(1)$	$\mathcal{R}(G) = \mathcal{O}((1/\phi)\log n)$,	w.h.p.	$\Omega(n \log n \log \Delta)$	[21, 14]
	$\mathcal{H}(G) = \mathcal{O}((1/\phi)\log n)$,	w.h.p.	$\mathcal{O}((1/\phi)\log^2 n)$	Thm. 8
Expanders	$\mathcal{R}(G) = \mathcal{O}(\log n)$,	w.h.p.	$\Theta(n \log n)$	[21, 14]
	$\mathcal{H}(G) = \mathcal{O}(\log n)$,	w.h.p.	$\mathcal{O}(\log^2 n)$	Thm. 8
Strong Expanders	$\mathcal{R}(G) = \log n + \ln n + o(\log n)$,	w.p. $1 - o(1)$	$\Theta(n \log n \log \Delta)$	Cor. 10
	$\mathcal{Q}(G) = \mathcal{O}(\log n)$,	w.h.p.	$\Theta(n \log \Delta)$	[6, 7]
	$\mathcal{P}(G) = \log n + \ln n + o(\log n)$,	w.p. $1 - o(1)$	$\mathcal{O}(\log^3 n)$	Thm. 9
Complete Graphs	$\mathcal{R}(G) = \log n + \ln n + o(\log n)$,	w.p. $1 - o(1)$	$\Theta(n \log^2 n)$	[23]
	$\mathcal{Q}(G) = \log n + \ln n + o(\log n)$,	w.p. $1 - o(1)$	$\Theta(n \log n)$	[11]
	$\mathcal{P}(G) = \log n + \ln n + o(\log n)$,	w.p. $1 - o(1)$	$\mathcal{O}(\log^3 n)$	Thm. 9
$G(n,p)$ with $p = \omega(\log n/n)$	$\mathcal{R}(G) = \log n + \ln n + o(\log n)$,	w.p. $1 - o(1)$	$\Theta(n \log n \log(pn))$	Cor. 10 & [12]
	$\mathcal{Q}(G) = \mathcal{O}(\log n)$,	w.h.p.	$\Theta(n \log(pn))$	[6]
	$\mathcal{P}(G) = \log n + \ln n + o(\log n)$,	w.p. $1 - o(1)$	$\mathcal{O}(\log^3 n)$	Cor. 12

Table 1 Comparison of the rumor spreading time and the required number of random bits for various topologies. By $\mathcal{R}(G)$ and $\mathcal{Q}(G)$ we denote the rumor spreading time of the fully random and the quasi-random push-protocols, respectively, on graph G. By $\mathcal{H}(G)$ and $\mathcal{P}(G)$ we denote the rumor spreading time of our hashing-based and PRG-based protocols. By δ and Δ we denote the minimum and maximum degrees of G, and ϕ denotes the conductance of G (see Section 3.2 for the definition of conductance). All the time bounds for $\mathcal{H}(G)$ listed above also hold for $\mathcal{P}(G)$.

be employed to show that there *exists* a protocol which needs only $\mathcal{O}(\log n)$ random bits to inform all nodes in the complete graph [15], no explicit construction of a protocol that uses less than $\mathcal{O}(n)$ random bits was known prior to this work.

1.1 Our Results

We present the first explicit rumor spreading protocols that use a sub-linear number of random bits in total to efficiently spread the rumor in a wide class of networks. We describe two protocols: one that uses hash functions, and one that uses pseudo-random generators (short: PRGs). If the protocols run for a number of t rounds, then they need $\mathcal{O}(t \cdot \log n)$ and $\mathcal{O}(t \cdot \log^2 n)$ random bits, respectively. We prove that for many standard graph topologies a logarithmic or polylogarithmic number of rounds suffice to broadcast the rumor w.h.p., so only a polylogarithmic number of random bits are consumed. In particular, using only a polylogarithmic number of random bits, our protocols are asymptotically as efficient as the best known protocol (i.e., the fully random one) on expanders and "strong" expanders (for the definition of strong expanders see the beginning of Section 4). For strong expanders, such as the complete graph and random graphs $G(n,p)$ with $p = \omega(\log n/n)$, our time bound matches the lower bound for regular graphs shown in [9] for the fully random protocol. We also prove a general upper bound of $\mathcal{O}((1/\phi)\log n)$ rounds, where ϕ is the conductance of the underlying graph. This bound is tight in the sense that there are graphs for which the diameter is at least $\Omega((1/\phi)\log n)$ [2]. The same upper bound was shown for the fully random push-protocol in [2, 14, 21]. For a more complete overview of our results and a comparison with previous results, see Table 1.

In our hashing-based protocol, nodes use *pairwise independent hash functions* (one for each round) to determine the neighbors to send the rumor to. The intuition is the following: In some round every informed node v establishes a communication link to a random neighbor

$X(v)$. For the efficiency of the protocol it is important that many of the random variables $X(v)$ are distinct, i.e., that the number of "message collisions" is small. In order to bound the number of collisions, we can use second-moment methods and thus rely on pairwise independence of the random variables $X(v)$, as opposed to complete independence. This is in spirit similar to the first application of pairwise independence to reduce the amount of randomness, namely Luby's derandomization of his parallel Minimum Independent Set algorithm [20]. In our PRG-based protocol, nodes employ Nisan's *pseudo-independent block generator* [22] with a different seed in every round. We assume that nodes have no initial IDs, so we combine our protocols with a mechanism to distribute IDs to all nodes. (Such a mechanism was already presented in [15], but in our new protocols the size of the IDs is much smaller.)

The analyses of both protocols deviates sometimes significantly from previous analyses of rumor spreading protocols that use full randomness. Since we are limited to pairwise independence or pseudo-independence, we cannot employ strong tail bounds such as Chernoff-type bounds.

Our protocols are local, i.e., no information about the graph topology is needed. Further, the fact that their analysis works for such a wide range of graphs indicates that the protocols are robust. While the protocols are not quite as simple as the fully random and the quasi-random protocols, the computation a node must perform in each round involves only a constant number of arithmetic operations in the hashing-based protocol, and $\mathcal{O}(\log n)$ operations in the PRG-based protocol. The randomness requirement of the latter protocol is also by a $(\log n)$-factor higher. The hashing-based protocol is asymptotically as efficient as the PRG-based protocol on all graphs that we consider. However, only for the PRG-based protocol the constant factor in our upper bound on the rumor spreading time for strong expander graphs matches the lower bound for the fully random protocol [9].

We assume the *standard adversary model*, which was also used in [6, 15]: In each round, every informed node u chooses an index $j \in \{1, \ldots, \deg(u)\}$, and sends a message to the j-th node in its adjacency list. No edge connection information is available to u other than its adjacency list; and the order of u's neighbors in this list is determined by an oblivious adversary (before the algorithm is executed).[2] For this model, it is known that any protocol for the complete graph that uses at most $b < \log n$ random bits ($b = 0$ for a deterministic algorithm) needs at least roughly $b + n/2^b$ rounds to inform all nodes [15].

1.2 Preliminaries

Throughout the paper, $G = (V, E)$ is a connected, undirected graph on n nodes. For each node $u \in V$, we let $N(u)$ denote the set of neighbors of u, and $\deg(u) := |N(u)|$ is the degree of u. By δ and Δ we denote the minimum and maximum degrees of G, respectively; if G is a regular graph, we denote its degree by d. For any node sets $S, T \subseteq V$, we define the edge set $E(S, T) := \{\{u, v\} \in E \mid u \in S \text{ and } v \in T\}$. The volume of S is $\mathrm{vol}(S) := \sum_{u \in S} \deg(u)$.

By I_t, for $t \geq 1$, we denote the set of informed nodes at the end of round t, and $U_t := V \setminus I_t$ is the set of uninformed nodes at that time. By I_0 and U_0 we denote the corresponding sets initially, before the algorithm is executed. We assume that $I_0 = \{s\}$ for some arbitrary initial node s. By $\log x$ we denote the binary logarithm of x.

[2] In fact, our results hold for a slightly stronger adversary: the adversary is allowed to choose a different ordering of the neighbors of a node for each round—but these orderings must be fixed before the algorithm starts.

For each $1 \leq i \leq m$, let X_i be a discrete uniform random variable with (finite) range R_i. We say that the sequence of random variables $X_1, \ldots, X_m$ is *pseudo-independent with parameter* ε, if for all $A_i \subseteq R_i$, $1 \leq i \leq m$,

$$\left| \mathbf{Pr}\left[X_1 \in A_1 \wedge \cdots \wedge X_k \in A_m \right] - \frac{|A_1 \times \cdots \times A_m|}{|R_1 \times \cdots \times R_m|} \right| \leq \varepsilon. \tag{1}$$

The sequence is *ε-approximate independent*, if equation (1) is true as long as all sets A_i, $1 \leq i \leq m$, have cardinality 1. Finally, the sequence is *ε-approximate k-wise independent*, if any k random variables in that sequence are ε-approximate independent.

We recall the following Chernoff bound which can be easily derived from the Chernoff bound for binomial random variables (cf. [8, Problem 3.6]).

▶ **Lemma 1.** *Fix any $0 < p < 1$ and let $X_1, \ldots, X_n$ be independent identical geometric random variables with $\mathbf{Pr}[X_i = k] = (1-p)^{k-1} \cdot p$, for every $k \geq 1$. Let $X = \sum_{i=1}^{n} X_i$ and $\mu = \mathbf{E}[X]$. Then it holds for all $\beta > 0$ that $\mathbf{Pr}[X \geq (1+\beta)\mu] \leq \mathrm{e}^{-n\beta^2/(2(1+\beta))}$.*

Due to space limitations, several proofs are omitted from this extended abstract.

2 Description of the Protocols

Both our protocols are of the same structure. We consider only T-round protocols, in which nodes send messages only for the first T rounds. In addition to T, each protocol is parameterized by n, the number of nodes in the graph, and c, a constant to adjust the error term. We assume that all nodes know n. (As long as the first node to receive the rumor knows some upper bound n' on n, the protocols work fine, but then the required amount of randomness is a function in n'.) Moreover, all nodes have access to a common random string s, the current round number, and the parameters c and T. (We make this assumption for presentational reasons. In practice, the first node can determine s, c, and T, and then send them together with the rumor. The current round number can also be sent together with the rumor, and updated in each round. From Table 1 it is immediate that the increase in message size incurred by this additional information is dominated by the length of s, i.e., the randomness requirements of the protocol.) Whenever a node sends the rumor to one of its neighbors in round t it also sends a message containing a unique string x that we call ID. A node is *uninformed* as long as it has not received a message. Once a node receives the first message, it becomes *informed* and from then on uses the ID of the received message as its own ID. Once a node becomes informed, it ignores all further incoming messages. If in some round an uninformed node receives multiple messages, then it only considers an arbitrary one and discards all others.

We assume that in a T-round protocol, the first node to receive the rumor (in round 0) also receives the ID 0. Now, suppose a node v receives its first message with ID x_v in round t'. (Recall that v discards all other messages it receives.) Then, in round $t > t'$ node v uses the values of t, s, as well as the ID x_v to determine two functions $f_t(s, x_v)$ (called the *link function*) and $g_t(x_v)$ (called the *ID function*). The function value of f_t modulo $\deg(v)$ determines to which neighbor v sends the rumor in round t, and $g_t(x_v)$ is the ID of the message sent by v to its neighbor in round t.

Both our protocols use the same ID function g_t, which is defined by $g_t(x) := x + 2^{t-1}$.

▶ **Claim 2.** *All messages of a T-round protocol with ID function $g_t(x) = x + 2^{t-1}$ have distinct IDs in $[2^T]$.*

Let us briefly describe the intuition why this ID function works. Consider any path $P = (u_0, u_1, \ldots, u_t)$ of length t that follows the "spread" of the rumor, i.e., if $u_{\ell-1} \neq u_\ell$, then $u_{\ell-1}$ sends the rumor to u_ℓ in round ℓ. Then, we associate with P a bitstring $S(P) = (s_1, \ldots, s_t)$ where $s_\ell = 1$ if $u_\ell \neq u_{\ell-1}$ and $s_\ell = 0$ otherwise. Note that $S(P) \neq S(P')$ for any two distinct paths P and P' that the rumor has followed in the first t rounds. Since additionally $S(P)$ is precisely the ID of the message that u_{t-1} sends to u_t in round t if $u_{t-1} \neq u_t$, it follows that all messages sent in round t have distinct IDs of length t whose last bit is 1.

For every node $v \in I_{t-1}$, let $X_t(v)$ be the random variable which assumes value $j \in \{1, \ldots, \deg(v)\}$ if in round t node v sends a message to its j-th neighbor (i.e., the j-th node in its adjacency list). If $v \notin I_{t-1}$, then define $X_t(v) = 0$. A protocol is *ε-approximate k-wise independent in round t*, if given any values of the random variables $X_{t'}(v)$, $t' < t$, $v \in V$, the sequence of random variables $X_t(v)$, $v \in I_{t-1}$, is ε-approximate k-wise independent. The protocol is *ε-approximate k-wise independent*, if it is so in every round. The protocol is *pseudo-independent with parameter ε in round t*, if given any values for the random variables $X_{t'}(v)$, $t' < t$, $v \in V$, the sequence of random variables $X_t(v)$, $v \in I_{t-1}$, is pseudo-independent with parameter ε.

In the following we present two T-round protocols: Our first protocol is based on pairwise independent hash functions. It is approximate pairwise independent and uses $\mathcal{O}\big(T \cdot (\log T + \log n)\big)$ random bits. Our second protocol is based on Nisan's PRG. It uses $\mathcal{O}(T \cdot \log^2 n)$ random bits, and is pseudo-independent.

2.1 The Hashing-Based Protocol

We present a simple protocol based on pairwise independent hash functions that achieves approximate pairwise independence. A family $\mathcal{H}$ of hash functions $h : [M] \to [N]$ is called *ε-approximate k-wise independent*, if for a randomly chosen function $h \in \mathcal{H}$ the sequence of hash values $h(x)$, $x \in [M]$, is ε-approximate k-wise independent. (In the case $\varepsilon = 0$, $\mathcal{H}$ is called k-wise independent.)

▶ **Claim 3.** *For all integer functions $R = R(n), M = M(n), N = N(n)$, there is an $\mathcal{O}(1/R)$-approximate pairwise independent family $\mathcal{H}_{M,N,R}$ of hash functions $h : [M] \to [N]$, such that each hash function in $\mathcal{H}_{M,N,R}$ can be described with $\mathcal{O}(\log R + \log\log M)$ bits.*

The construction of the hash class $\mathcal{H}_{M,N,R}$ is standard: Every function in $\mathcal{H}_{M,N,R}$ has the same form $h_{a,b,p}(x) = (ax + b) \bmod p \bmod N$, where p is a prime in $\{M', 2M'\}$, $M' = \lceil R \cdot \log M \rceil$, and $a, b \in [p]$. The random linear functions over $[p]$ yield pairwise independence over $[p]$ for all pairs of keys that are not in the same congruence class modulo p. Since p is a random prime for a randomly chosen hash function, the probability that two keys are congruent modulo p is small, and we obtain approximate pairwise independence.

Our protocol uses a sequence of randomly chosen hash functions. More precisely, the random string s used by the protocol is a sequence of T hash functions, i.e., $s = (h_1, \ldots, h_T)$, where $h_i \in \mathcal{H}_{2^T, n^c, n^{3c}}$ is chosen uniformly (and independently) at random. The protocol uses the link function $f_t\big(s, x_v\big) = h_t(x_v)$. That is, any node $v \in I_{t-1}$ with ID x_v sends the rumor in round t to the $(\ell_t + 1)$-th node in its neighbor list, where $\ell_t = \big(h_t(x_v)\big) \bmod \deg(v)$. (Recall that it also sends a message with the ID $g_t(x_v)$ along with the rumor.)

▶ **Lemma 4.** *The hashing-based T-round protocol is $\mathcal{O}(1/n^c)$-approximate pairwise independent and uses $\mathcal{O}(T \cdot (\log T + \log n))$ random bits.*

Proof. By Claim 3, a hash function $h_i \in \mathcal{H}_{2^T, n^c, n^{3c}}$ can be described with $\mathcal{O}(\log T + \log n)$ random bits. Hence, the protocol uses T times that many random bits.

Now fix some round number $t \leq T$ and some hash functions $h_1, \ldots, h_{t-1} \in \mathcal{H}_{2^T,n^c,n^{3c}}$. Then the execution of the protocol during the first $t-1$ rounds is uniquely determined by the choice of $h_1, \ldots, h_{t-1}$. Now, let $u, v \in I_{t-1}$ be distinct nodes and fix arbitrary $y_u \in [\deg(u)]$ and $y_v \in [\deg(v)]$. Suppose that u and v obtained IDs x_u and x_v, respectively, when they received their first messages. By Claim 2, $x_u \neq x_v$. Then, even though the execution of the protocol during the first $t-1$ rounds is fixed (and the values of x_u and x_v may depend on $h_1, \ldots, h_{t-1}$), $h_t(x_u)$ and $h_t(x_v)$ are $\mathcal{O}(1/n^{3c})$-approximate pairwise independent random variables with range $[n^c]$. Thus, for every pair $(z_u, z_v) \in [n^c]^2$ the probability that $h_t(x_u) = z_u$ and $h_t(x_v) = z_v$ is at most $1/n^{2c} + \mathcal{O}(1/n^{3c}) = (1 + \mathcal{O}(1/n^c))/n^{2c}$. Hence,

$$\begin{aligned}
&\mathbf{Pr}\left[\, h_t(x_u) \bmod \deg(u) = y_u \wedge h_t(x_v) \bmod \deg(v) = y_v \,\right] \\
&\leq \left(1 + \mathcal{O}\left(\frac{1}{n^c}\right)\right) \cdot \frac{1}{n^{2c}} \cdot \lceil n^c/\deg(u) \rceil \cdot \lceil n^c/\deg(v) \rceil \\
&\leq \left(1 + \mathcal{O}\left(\frac{1}{n^c}\right)\right) \cdot \left(\frac{1}{\deg(u)} + \frac{1}{n^c}\right) \cdot \left(\frac{1}{\deg(v)} + \frac{1}{n^c}\right) = \frac{1}{\deg(u) \cdot \deg(v)} + \mathcal{O}\left(\frac{1}{n^c}\right).
\end{aligned}$$

A similar calculation gives a lower bound of $\frac{1}{\deg(u)\deg(v)} - \mathcal{O}(\frac{1}{n^c})$ on the above probability. ◀

In order to select their random neighbors, nodes have to randomly choose hash functions from the class $\mathcal{H}_{2^T,n^c,n^{3c}}$. Evaluating a hash function involves only a few integer arithmetic operations with integers of value at most 2^T. However, in order to select a random hash function, one also has to select a random prime of logarithmic length (in n). This can be avoided, though, by using hash functions in $\mathcal{H}_{2^T,n^c,R}$, where $R = \max\{n^{3c}, 2^T\}$, as in this case it can be shown that the hash functions do not need to use *random* primes. Doing this increases the total number of random bits used by the protocol to $\mathcal{O}\big(T \cdot (T + \log n)\big)$. For most of the graph topologies analyzed in this paper, $T = \mathcal{O}(\log n)$, so in this case sampling random primes can be avoided without affecting the randomness requirement.

2.2 The PRG-Based Protocol

Let m, ℓ, and k be positive integers and let $\varepsilon > 0$. Let $B : \{0,1\}^m \to \big(\{0,1\}^\ell\big)^k$ be some mapping. For $0 \leq i < k$ and $s \in \{0,1\}^m$, we define $B_i(s)$ to be the projection of $B(s)$ to the $(i+1)$-th component. That is, if $B(s) = y_0 y_1 \ldots y_{k-1}$, where each $y_j \in \{0,1\}^\ell$ is a block of length ℓ, then $B_i(s) = y_i$. The mapping B is a *pseudo-independent block generator with parameter* ε, if for a randomly chosen *seed* $w \in \{0,1\}^m$ the random variables $B_0(w), \ldots, B_{k-1}(w)$ are pseudo-independent with parameter ε.

▶ **Theorem 5** ([22]). *There is a constant $\alpha > 0$ such that for any integers ℓ and $k \leq \ell$ there is a pseudo-independent block generator $B^{(\ell,k)} : \{0,1\}^{\alpha \cdot \ell \cdot k} \to \big(\{0,1\}^\ell\big)^{2^k}$ with parameter $2^{-\ell}$.*

The block generator is based on pairwise independent hash functions. In particular the random seed is an ℓ-bit string x and a sequence of k hash functions $h_1, \ldots, h_k$ from a family of pairwise independent hash functions with universe and range of size roughly 2^ℓ. In order to determine a random value $B_i(x)$ it suffices to evaluate the composition of up to k hash functions at point x. (Hence, it is not required for nodes to generate the entire pseudo-random string.)

Let α be the constant from Theorem 5 and $\ell = \lceil c \cdot \log n \rceil$. Our T-round protocol uses a random string $s = \big((p_1, w_1), \ldots, (p_T, w_T)\big)$, where each $w_i \in \{0,1\}^{\alpha \cdot \ell^2}$ and p_i is a random prime in $\{2^{\ell-1}, \ldots, 2^\ell\}$. As previously, nodes send messages in order to distribute IDs using the ID function $g_t(x_v)$. If a node $v \in I_{t-1}$ has ID x_v, it uses the link function

$f_t(s, x_v) = B^{(\ell,\ell)}_{x_v \bmod p_t}(w_t)$ to determine to which neighbor to send the rumor to in round t. That is, if prior to round t node v receives its first message with ID x_v, then in round t it determines $b = x_v \bmod p_t$ and looks up the $(b+1)$-th block of the random string determined by the block-generator $B^{(\ell,\ell)}$ with seed w_t. (This random string consists of 2^ℓ blocks of length ℓ.)

▶ **Lemma 6.** *The PRG-based T-round protocol is pseudo-independent with parameter $\mathcal{O}(T/n^{c-2})$ and uses $\mathcal{O}(T \cdot \log^2 n)$ random bits.*

As in the hashing based protocol, one can avoid generating a random prime for each round by choosing $\ell = \max\{T, \lceil c \cdot \log n \rceil\}$ and defining $f_t(s, x_v) = B^{(\ell,\ell)}_{x_v}(w_t)$ (thus, nodes do not consider their IDs modulo a prime). This way, the protocol needs $\mathcal{O}\big(T(T^2 + \log^2 n)\big)$ random bits, which is no different than before as long as $T = \mathcal{O}(\log n)$.

3 Analysis of the Hashing-Based Protocol

In Section 3.1 we study the progress achieved in a single round of an approximate pairwise independent protocol. Then in Section 3.2, we use this result to derive a general upper bound for the hashing-based protocol in terms of graph conductance.

3.1 Analysis of a Single Round

The next lemma provides lower bounds on the number of nodes informed in a single round of an ε-approximate pairwise independent protocol, for a sufficiently small ε. Specifically, it counts the nodes informed by rumor transmissions along the edges in a given subset F of the set $E(I_t, U_t)$ of edges between informed and uninformed nodes. Note that the expected number of such transmissions in the fully random protocol is $\sum_{\{u,v\} \in F:\, u \in I_t, v \in U_t} 1/\deg(u)$, which is at least $|F|/\Delta$ and at most $|F|/\delta$. For an ε-approximate pairwise independent protocol, the expected number of such transmissions is different by at most $\varepsilon\Delta \cdot |F|$. Let X_u, for each $u \in I_t$, denote the neighbor $v \in N(u)$ that u chooses in round $t+1$. By the law of total probability, for any node $u' \in I_t \setminus \{u\}$,

$$\begin{aligned}\mathbf{Pr}\,[\,X_u = v\,] &= \sum_{v' \in N(u')} \mathbf{Pr}\,[\,X_u = v \wedge X_{u'} = v'\,] \le |N(u')| \cdot \left(\frac{1}{\deg(u) \cdot \deg(u')} + \varepsilon\right) \\ &\le 1/\deg(u) + \varepsilon\Delta,\end{aligned}$$

and similarly, $\mathbf{Pr}\,[\,X_u = v\,] \ge 1/\deg(u) - \varepsilon\Delta$.

▶ **Lemma 7.** *Consider an ε-approximate pairwise independent protocol with $\varepsilon = o(1/n^3)$. Fix a round $0 \le t < T$ and the set I_t of informed nodes before round $t+1$ begins. Fix also an arbitrary set of edges $F \subseteq E(I_t, U_t)$. Let J be the set of nodes that become informed in round $t+1$ if we consider only transmissions of the rumor along the edges in F.*

(a) $\mathbf{Pr}\,[\,|J| \ge 1\,] \ge (1 - o(1))\,\dfrac{|F|/\Delta}{2|F|/\delta + 6}$.

(b) If $|F| \ge 16\Delta$, *then* $\mathbf{Pr}\left[\,|J| \ge \dfrac{1}{19} \cdot \dfrac{\delta^2}{\Delta^2} \cdot \dfrac{|F|}{\Delta}\,\right] \ge \dfrac{1}{2} - o(1)$.

(c) For any $v \in U_t$, let $\gamma_v := |\{u \in V : \{u, v\} \in F\}|$ be the number of edges in F that are incident to v. If $\sum_{v \in V} \gamma_v^2 = o(\Delta \cdot |F|)$, and $|F| = \omega(\Delta)$, and $\Delta/\delta = 1 + o(1)$, then

$$\mathbf{Pr}\left[\,|J| \ge (1 - o(1)) \cdot \frac{|F|}{\Delta}\,\right] \ge 1 - o(1).$$

▶ Remark. We will use Lemma 7 in the analysis of the PRG-based protocol as well; in fact, (c) is only used there. Recall from Lemma 6 that the PRG-based protocol is pseudo-independent with parameter $\varepsilon = \mathcal{O}(T/n^{c-2})$, and thus it is also ε-approximate pairwise independent.

We now give an outline of the proof of Lemma 7. For any uninformed node $v \in U_t$, let Z_v denote the number of rumor transmissions to v through edges in F in round $t+1$. To prove (a) we bound the probability that $\sum_v Z_v = 0$, which is the same as the probability that $|J| = 0$, using Chebyshev's inequality. Note that we cannot use stronger concentration tools, such as Chernoff bounds, since we only have (approximate) pairwise independence among the choices of different nodes in a round. In general, $\sum_v Z_v \geq |J|$, because a node may receive the rumor more than once in a round. Thus we cannot prove (b) and (c) just by showing a lower bound on $\sum_v Z_v$. In addition to lower-bounding $\sum_v Z_v$, we also upper-bound $\sum_v Z_v^2$. For the latter bound we use Markov's inequality (we cannot apply Chebyshev's inequality, as 4-wise independence among the node's choices is needed for that). Then we apply Cauchy-Schwartz's inequality to lower-bound $|J| = \sum_v \mathbf{1}_{Z_v>0}$ by $\left(\sum_v Z_v\right)^2 / \sum_v Z_v^2$.

3.2 An Upper Bound in Terms of Conductance

Let G be an arbitrary graph and let $S \subseteq V$ be any set of size $0 < |S| < n$. The conductance of S is defined as $\phi(S) = \frac{|E(S,V\setminus S)|}{\min\{\mathrm{vol}(S),\mathrm{vol}(V\setminus S)\}}$. The conductance of G is defined as the minimum conductance over all sets S,

$$\phi(G) = \min_{S\subseteq V, 0<|S|<n} \Phi(S) = \min_{S\subseteq V, 0<|S|<n} \frac{|E(S, V\setminus S)|}{\min\{\mathrm{vol}(S), \mathrm{vol}(V\setminus S)\}}.$$

Note that the conductance of a d-regular graph G is $\phi(G) = \min_{S\subseteq V, 0<|S|<n} \frac{|E(S,V\setminus S)|}{d\cdot\min\{|S|, n-|S|\}}$.

▶ **Theorem 8.** *For any graph with conductance ϕ and $\Delta/\delta = \mathcal{O}(1)$, the hashing-based protocol informs all nodes in $\mathcal{O}((1/\phi)\log n)$ rounds w.h.p. using $\mathcal{O}((1/\phi)\log^2 n)$ random bits.*

Proof. We choose the parameters of the hashing-based protocol to be $T = \Theta((1/\phi)\log n)$ and $c > 3$. From Lemma 4 then it follows that the protocol is $o(1/n^3)$-approximate pairwise independent, and the total number of random bits used is $\mathcal{O}(T\cdot(\log T + \log n)) = \mathcal{O}((1/\phi)\log^2 n)$, since $1/\phi = \mathcal{O}(n^2)$ in any connected graph.

The proof is divided into four phases according to the number of informed nodes $|I_t|$.

Phase 1: $1 \leq |I_t| \leq 16(\Delta/\delta)\cdot(1/\phi)$. This phase is divided into several subphases. For every $1 \leq i \leq \log(16(\Delta/\delta)\cdot(1/\phi))$, subphase i begins when the number of informed nodes is at least 2^{i-1} and ends when this number is at least 2^i. Assume that we are at the beginning of the i-th subphase. Fix an arbitrary round t of the i-th subphase and the set of informed nodes I_t; thus, $2^{i-1} \leq |I_t| < 2^i$. We consider the number of nodes that become informed in round $t+1$. Applying Lemma 7(a) with $F = E(I_t, U_t)$ gives

$$\mathbf{Pr}\left[\,|I_{t+1}\setminus I_t| \geq 1\,\right] \geq (1-o(1))\frac{|E(I_t,U_t)|/\Delta}{2|E(I_t,U_t)|/\delta + 6}. \tag{2}$$

Suppose first that $|E(I_t,U_t)|/\delta \geq 6$. Then, the above probability is at least

$$(1-o(1))\cdot(\delta/\Delta)\cdot(1/3) \geq (\delta/\Delta)^2\cdot(1/49)\cdot|I_t|\cdot\phi =: p,$$

where the inequality follows from the upper bound on $|I_t|$. On the other hand, if $|E(I_t,U_t)|/\delta \leq 6$, then the probability in equation (2) is at least

$$(1-o(1))\cdot|E(I_t,U_t)|/(18\Delta) \geq (1-o(1))\cdot\phi\,\delta\,|I_t|/(18\Delta) \geq p.$$

Therefore, the expected time to increase $|I_t|$ from 2^{i-1} to 2^i is at most

$$2^{i-1} \cdot \frac{1}{p} \leq 2^{i-1} \cdot \frac{1}{(1/49)\,(\delta/\Delta)^2 \cdot \phi\, 2^{i-1}} = 49 \cdot (\Delta/\delta)^2 \cdot (1/\phi) =: \tau.$$

By Markov's inequality,

$$\mathbf{Pr}\left[\, |I_{t+2\tau}| \leq 2^i \;\mid\; |I_t| \geq 2^{i-1} \,\right] \leq 1/2.$$

Hence the time to complete Phase 1 can be upper bounded by $\tau = \mathcal{O}((1/\phi))$ multiplied with the sum of $\log(16(\Delta/\delta) \cdot (1/\phi)) = \mathcal{O}(\log n)$ independent geometric random variables each with parameter $1/2$. Applying a Chernoff bound for the sum of independent geometric random variables (Lemma 1) yields that the number of rounds required for Phase 1 is at most $\mathcal{O}((1/\phi) \cdot \log n)$ w.h.p.

Phase 2: $16(\Delta/\delta) \cdot (1/\phi) \leq |I_t| \leq n/2$. Fix a round t and the set of informed nodes I_t. We apply Lemma 7(b), with $F = E(I_t, U_t)$. Note that the precondition $|F| \geq 16\Delta$ is satisfied, as

$$|F| = |E(I_t, U_t)| \geq \phi \cdot \delta \cdot |I_t| \geq \phi \cdot \delta \cdot 16(\Delta/\delta) \cdot (1/\phi) = 16\Delta.$$

Hence we conclude from Lemma 7(b),

$$\mathbf{Pr}\left[\, |I_{t+1} \setminus I_t| \geq \frac{1}{19} \cdot \frac{\delta^3}{\Delta^3} \cdot \phi \cdot |I_t| \,\right] \geq \frac{1}{2} - o(1),$$

and thus, with probability $1/2 - o(1)$, $|I_{t+1}| \geq \left(1 + \frac{1}{19} \cdot \frac{\delta^3}{\Delta^3} \cdot \phi\right) \cdot |I_t|$. So, the number of rounds until we have $|I_t| \leq n/2$ can be upper bounded by the sum of $\log_{1+\frac{1}{19}\cdot\frac{\delta^3}{\Delta^3}\cdot\phi}(n/2) = \mathcal{O}((1/\phi)\log n)$ independent geometric random variables with parameters $1/2 - o(1)$. Using again the Chernoff bound in Lemma 1 we obtain that Phase 2 is completed within at most $\mathcal{O}((1/\phi)\log n)$ rounds w.h.p.

Phase 3: $n/2 \leq |I_t| \leq n - 16(\Delta/\delta) \cdot (1/\phi)$. The analysis is the same as in Phase 2 with the roles of I_t and U_t switched.

Phase 4: $n - 16(\Delta/\delta) \cdot (1/\phi) \leq |I_t| \leq n$. Again, the analysis is the same as in Phase 1 with the roles of I_t and U_t switched.

Since each of the four phases requires only $\mathcal{O}((1/\phi) \cdot \log n)$ rounds w.h.p., the result follows by applying the union bound. ◀

4 Analysis of the PRG-Based Protocol

We now consider graph families with strong expansion properties. We prove that by increasing the number of random bits slightly, from $\mathcal{O}(\log^2 n)$ to $\mathcal{O}(\log^3 n)$, we can obtain precise time bounds that are comparable to the ones for the fully random protocol.

We describe a condition that implies such tight bounds, in terms of the following version of conductance (see, e.g., [18]),

$$\tilde{\phi}(G) := \min_{S \subseteq V, 0<|S|<|V|} \frac{|E(S, V \setminus S)| \cdot \mathrm{vol}(V)}{\mathrm{vol}(S) \cdot \mathrm{vol}(V \setminus S)}.$$

This definition is slightly different than the one given in Section 3.2, but it is easy to verify that $\phi(G) \leq \tilde{\phi}(G) \leq 2 \cdot \phi(G)$.

The following theorem concerns so-called *strong expanders*, which are almost-regular graphs for which the conductance $\tilde{\phi}(G)$ tends to one.

▶ **Theorem 9.** *For any graph with* $\Delta/\delta = 1 + o(1)$ *and* $\tilde{\phi} \geq 1 - o(1)$*, the PRG-based protocol informs all nodes in* $\log n + \ln n + o(\log n)$ *rounds with probability* $1 - o(1)$*, using* $\mathcal{O}(\log^3 n)$ *random bits in total.*

The proof of Theorem 9 is similar to that of Theorem 8. We consider different phases according to the size of I_t and apply Lemma 7(c) to lower bound the increase of the number of informed nodes.

It was shown in [9, Theorem 1 & Lemma 2] that on any d-regular graph with $d = \omega(1)$, the fully random protocol requires at least $\log n + \ln n - o(\log n)$ rounds to spread the rumor to all n nodes. We observe the following simple corollary of Theorem 9.

▶ **Corollary 10.** *Under the same assumptions as in Theorem 9, all nodes are informed by the fully random protocol within* $\log n + \ln n + o(\log n)$ *rounds with probability* $1 - o(1)$*.*

Theorem 9 can be used to obtain tight bounds for several interesting graph families. For that, we consider the algebraic expansion of graphs. For any graph $G = (V, E)$, let M be the normalized adjacency matrix of G, i.e., $M_{i,j} = 1/\sqrt{\deg(i)\deg(j)}$ if $\{i, j\} \in E$ and $M_{i,j} = 0$ otherwise. Moreover, let $\lambda_2 = \lambda_2(G)$ be the second largest eigenvalue of M. Since M is real and symmetric, λ_2 is a real number. Also, since $\tilde{\phi}(G) \geq 1 - \lambda_2$ [18, Theorem 5.3], we can apply Theorem 9 to obtain the following result.

▶ **Corollary 11.** *For any graph with* $\lambda_2 = o(1)$ *and* $\Delta/\delta = 1 + o(1)$*, the PRG-based protocol informs every node in* $\log n + \ln n + o(\log n)$ *rounds with probability* $1 - o(1)$*, using* $\mathcal{O}(\log^3 n)$ *random bits in total.*

Notice that this corollary can be applied to regular graphs. In particular, d-regular Ramanujan graphs [19] satisfy the preconditions of the corollary. Moreover, we can use Corollary 11 to obtain a time bound for certain families of random graphs.

▶ **Corollary 12.** *In the* $G(n, p)$ *random graph with* $p = \omega(\log n/n)$*, the PRG-based protocol informs every node in* $\log n + \ln n + o(\log n)$ *rounds with probability* $1 - o(1)$*, using* $\mathcal{O}(\log^3 n)$ *random bits in total.*

Proof. Since $p = \omega(\log n/n)$, we have $\lambda_2 = o(1)$ by [3, Theorem 1.2], and $\Delta/\delta = 1 + o(1)$ by a Chernoff bound. Applying Corollary 11 then yields the claim. ◀

References

1 S. Boyd, A. Ghosh, B. Prabhakar, and D. Shah. Randomized gossip algorithms. *IEEE Transactions on Information Theory and IEEE/ACM Transactions on Networking*, 52(6):2508–2530, 2006.

2 F. Chierichetti, S. Lattanzi, and A. Panconesi. Almost tight bounds on rumour spreading with conductance. In *42nd ACM Symposium on Theory of Computing (STOC'10)*, pages 399–408, 2010.

3 A. Coja-Oghlan. On the Laplacian eigenvalues of $G_{n,p}$. *Combinatorics, Probability & Computing*, 16(6):923–946, 2007.

4 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 *6th ACM Symposium on Principles of Distributed Computing (PODC'87)*, pages 1–12, 1987.

5 B. Doerr and M. Fouz. A time-randomness tradeoff for quasi-random rumour spreading. *Electronic Notes in Discrete Mathematics*, 34:335–339, 2009.

6 B. Doerr, T. Friedrich, and T. Sauerwald. Quasirandom rumor spreading. In *19th ACM-SIAM Symposium on Discrete Algorithms (SODA'08)*, pages 773–781, 2008.

7 B. Doerr, T. Friedrich, and T. Sauerwald. Quasirandom rumor spreading: Expanders, push vs. pull and robustness. In *36th International Colloquium on Automata, Languages, and Programming (ICALP'09)*, pages 366–377, 2009.

8 D. Dubhashi and A. Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009.

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

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

11 N. Fountoulakis and A. Huber. Quasirandom rumor spreading on the complete graph is as fast as randomized rumor spreading. *SIAM Journal on Discrete Mathematics*, 23(4):1964–1991, 2009.

12 N. Fountoulakis, A. Huber, and K. Panagiotou. Reliable broadcasting in random networks and the effect of density. In *29th IEEE Conference on Computer Communications (INFOCOM'10)*, pages 2552–2560, 2010.

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

14 G. Giakkoupis. Tight bounds for rumor spreading in graphs of a given conductance. In *28th International Symposium on Theoretical Aspects of Computer Science (STACS'11)*, pages 57–68, 2011.

15 G. Giakkoupis and P. Woelfel. On the randomness requirements of rumor spreading. In *22nd ACM-SIAM Symposium on Discrete Algorithms (SODA'11)*, pages 449–461, 2011.

16 M. Harchol-Balter, F. T. Leighton, and D. Lewin. Resource discovery in distributed networks. In *18th ACM Symposium on Principles of Distributed Computing (PODC'99)*, pages 229–237, 1999.

17 R. Karp, C. Schindelhauer, S. Shenker, and B. Vöcking. Randomized rumor spreading. In *41st IEEE Symposium on Foundations of Computer Science (FOCS'00)*, pages 565–574, 2000.

18 L. Lovász. Random walks on graphs: A survey. *Combinatorics, Paul Erdös is Eighty*, 2:1–46, 1993.

19 A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan graphs. *Combinatorica*, 8(3):261–277, 1988.

20 M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

21 D. Mosk-Aoyama and D. Shah. Fast distributed algorithms for computing separable functions. *IEEE Transactions on Information Theory*, 54(7):2997–3007, 2008.

22 N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

23 B. Pittel. On spreading a rumor. *SIAM Journal on Applied Mathematics*, 47(1):213–223, 1987.

24 R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *15th IFIP International Conference on Distributed Systems Platforms (Middleware)*, pages 55–70, 1998.

Lower Bounds on the Complexity of MSO_1 Model-Checking

Robert Ganian[1], Petr Hliněný[1], Alexander Langer[2], Jan Obdržálek[1], Peter Rossmanith[2], and Somnath Sikdar[2]

1 **Faculty of Informatics, Masaryk University, Brno, Czech Republic***
{xganian1,hlineny,obdrzalek}@fi.muni.cz
2 **Theoretical Computer Science, RWTH Aachen University, Germany** †
{langer,rossmani,sikdar}@cs.rwth-aachen.de

Abstract

One of the most important algorithmic meta-theorems is a famous result by Courcelle, which states that any graph problem definable in monadic second-order logic with edge-set quantifications (MSO_2) is decidable in linear time on any class of graphs of bounded tree-width. In the parlance of parameterized complexity, this means that MSO_2 model-checking is fixed-parameter tractable with respect to the tree-width as parameter. Recently, Kreutzer and Tazari [13] proved a corresponding complexity lower-bound—that MSO_2 model-checking is not even in XP wrt. the formula size as parameter for graph classes that are subgraph-closed and whose tree-width is poly-logarithmically unbounded. Of course, this is not an unconditional result but holds modulo a certain complexity-theoretic assumption, namely, the Exponential Time Hypothesis (ETH).

In this paper we present a closely related result. We show that even MSO_1 model-checking with a fixed set of vertex labels, but without edge-set quantifications, is not in XP wrt. the formula size as parameter for graph classes which are subgraph-closed and whose tree-width is poly-logarithmically unbounded unless the non-uniform ETH fails. In comparison to Kreutzer and Tazari, (1) we use a stronger prerequisite, namely non-uniform instead of uniform ETH, to avoid the effectiveness assumption and the construction of certain obstructions used in their proofs; and (2) we assume a different set of problems to be efficiently decidable, namely MSO_1-definable properties on vertex labeled graphs instead of MSO_2-definable properties on unlabeled graphs.

Our result has an interesting consequence in the realm of digraph width measures: Strengthening the recent result [8], we show that no subdigraph-monotone measure can be algorithmically useful, unless it is within a poly-logarithmic factor of (undirected) tree-width.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Monadic Second-Order Logic, Treewidth, Lower Bounds, Exponential Time Hypothesis, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.326

1 Introduction

A famous result by Courcelle, proved in 1990, states that any graph property definable in monadic second-order logic with quantification over vertex- and edge-sets (MSO_2) can be decided in linear time on any class of graphs of bounded tree-width [2]. This result has a strong significance. As MSO_2 logic can express many interesting graph properties, we immediately

* All the three authors have been supported by the Czech Science Foundation, project P202/11/0196.
† Supported by Deutsche Forschungsgemeinschaft, project RO 927/9.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 326–337

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

get linear-time algorithms for important NP-hard problems, such as Hamiltonian Cycle, Vertex Cover, and 3-Colorability, on graphs of bounded tree-width. Such a result is called an *algorithmic meta-theorem*, and many other algorithmic meta-theorems have since appeared for other classes of graphs—see e.g. [9, 11] for a good survey.

As can be seen, Courcelle's theorem is a fast and relatively easy way of establishing that a problem can be solved efficiently on graphs of bounded tree-width. However, one may ask how far this result could be generalized. That is, is there a graph class of unbounded tree-width such that MSO_2 model-checking remains tractable on this class? Considering how important this question is for theoretical understanding of what makes some problems on certain graph classes hard, it is surprising that until recently there has not been much research in this direction.

The first result, by Kreutzer, providing a "lower bound" to Courcelle's theorem appeared in [12]. In that paper, Kreutzer used the following version of "unbounding" the tree-width of a graph class:

▶ **Definition 1** (Kreutzer and Tazari [12, 13])**.** The tree-width of a class $\mathcal{C}$ of graphs is strongly unbounded by a function $f\colon \mathbb{N} \to \mathbb{N}$ if there is an $\epsilon < 1$ and a polynomial $p(x)$ s.t. for all $n \in \mathbb{N}$ there is a graph $G_n \in \mathcal{C}$ with the following properties:

i) the tree-width of G_n is between n and $p(n)$ and is greater than $f(|G_n|)$, and
ii) given n, the graph G_n can be constructed in time 2^{n^ϵ}.

The degree of the polynomial p is called the *gap-degree* of $\mathcal{C}$ (with respect to f). The tree-width of $\mathcal{C}$ is *strongly unbounded poly-logarithmically* if it is strongly unbounded by $\log^c n$, for all $c \geq 1$.

In other words, the tree-width of $\mathcal{C}$ is *strongly unbounded* means that (i) there are no big gaps between the tree-width of witness graphs (those that certify that the tree-width of n-vertex graphs in $\mathcal{C}$ is greater than $f(n)$), and (ii) we can compute such witnesses effectively—in sub-exponential time wrt. n.

The main result of [12] is the following theorem (we postpone formal definitions to Sections 2 and 3): Let Γ be a fixed set of colors, and $\mathcal{C}$ be a class of graphs such that (1) the tree-width of $\mathcal{C}$ is strongly unbounded poly-logarithmically; (2) $\mathcal{C}$ is closed under Γ-colorings (i.e., if $G \in \mathcal{C}$ and G' is obtained from G by coloring some vertices or edges by colors from Γ, then $G' \in \mathcal{C}$); and, (3) $\mathcal{C}$ is constructible (i.e., given a witness graph in $\mathcal{C}$, a certain substructure can be computed in polynomial time). Then $\mathrm{MC}(\mathrm{MSO}_2\text{-}\Gamma, \mathcal{C})$, the MSO_2 model-checking problem on $\mathcal{C}$ with colors from Γ, is not in XP (and hence not in FPT—see Section 2 for a definition of these complexity classes), unless all problems in the polynomial-time hierarchy can be solved in sub-exponential time. This would, of course, mean that the Exponential-Time Hypothesis (ETH) fails. The results of [12] have been improved by Kreutzer and Tazari in [14], where the constructibility requirement (3) was dropped.

A further improvement by the same authors appeared in [13]. The main result in [13] can be stated as follows: Let $\mathcal{C}$ be a class of graphs such that (1) the tree-width of $\mathcal{C}$ is strongly unbounded poly-logarithmically; and (2′) $\mathcal{C}$ is closed under taking subgraphs, i.e. $G \in \mathcal{C}$ and $H \subseteq G$ implies $H \in \mathcal{C}$. Then $\mathrm{MC}(\mathrm{MSO}_2, \mathcal{C})$, the (ordinary) MSO_2 model-checking problem on $\mathcal{C}$, is not in XP unless all problems in the polynomial-time hierarchy can be solved in sub-exponential time. Note that (2′), to be closed under subgraphs, is a strictly weaker condition than (2), to be closed under Γ-colorings (of edges, too).

Our results

In this paper we prove a result closely related to Kreutzer–Tazari's [12, 14, 13] but for MSO_1 logic with a fixed set of vertex labels. The role of *vertex labels* in our paper is similar to that of colors in [12, 14], but weaker in the sense that the labels are not assigned to edges.[1] In contrast to the work by Kreutzer and Tazari, we assume a different set of problems—those expressible by MSO_1-L on graphs with vertex labels from a fixed finite set L—to be efficiently solvable on a graph class in order to derive an analogous conclusion.

Before stating our main result, we mention one more fact. There exist classes $\mathcal{C}$ of L-labeled graphs of unbounded tree-width on which $\mathrm{MC}(MSO_1\text{-}L, \mathcal{C})$, the MSO_1 model-checking problem on $\mathcal{C}$, is polynomial time solvable, e.g. classes of bounded clique-width or rank-width. But it is important to realize that these classes are *not* closed under taking subgraphs. Our main result then reads—cf. Section 4:

▶ **Theorem 2** (reformulated as Theorem 12). *Assume a (suitable but fixed) finite label set L, and a graph class $\mathcal{G}$ satisfying the following two properties:*

a) $\mathcal{G}$ is closed under taking subgraphs,
b) the tree-width of $\mathcal{G}$ is densely unbounded poly-logarithmically (see Def. 8).

Then $\mathrm{MC}(MSO_1\text{-}L, \mathcal{G}^L)$*, the* MSO_1*-L model-checking problem on all L-vertex-labeled graphs from $\mathcal{G}^L$, is* not *in* XP *unless the non-uniform Exponential-Time Hypothesis fails.*

Our general approach follows that by Kreutzer and Tazari in [12, 14, 13] but differs from theirs in three main ways:

I) Kreutzer and Tazari require witnesses as in (ii) of Definition 1 of [13] to be computable effectively in their proofs. It is unclear how this can be done and hence they simply add this as a natural requirement. Furthermore, the construction of certain obstructions (grid-like minors) used in their proof requires an involved machinery [14]. We adopt a different position and avoid (note our "densely unbounded" vs. "strongly unbounded") both aspects by using a stronger complexity-theoretic assumption, namely the non-uniform ETH instead of the ordinary ETH. In this way, we can get the obstructions as *advice* "for free." This makes our proof shorter and exhibits its structure more clearly.

II) Our result applies to MSO_1-L model-checking on *L-vertex-labeled graphs*, while the result of [13] applies to MSO_2 over unlabeled graphs. There are problems that can be expressed in MSO_1-L and not in MSO_2 and vice versa (take Red-Blue Dominating Set vs. Hamiltonian Cycle, for instance). If, however, the set of labels L is fixed for both, MSO_1-L has much weaker expressive power than MSO_2-L due to missing edge-set quantifications (see Section 2). In particular, note that many of the existing algorithmic meta-theorems (e.g. [2, 4]) that deal with MSO-definable properties handle unlabeled as well as (vertex-)labeled inputs with equal ease. However, extending e.g. the results of [4] from MSO_1-L to MSO_2 is not possible unless EXP = NEXP.

III) Finally, because of the free advice, our proof does not need technically involved machinery such as the simulation of a run of a Turing machine encoded in graphs [13].

Theorem 2 raises the open question whether poly-logarithmically unbounded tree-width along with closure under subgraphs is a strong enough condition for even the *bare* MSO_1 *model-checking to be intractable* (modulo appropriate complexity-theoretic assumptions).

[1] The reason we use the term labels and not colors is to be able to clearly distinguish between vertex-labeled graphs and the colored graphs used in [12, 14], where colors are assigned to edges and vertices.

If we assume that the label set L is "unbounded" we obtain an even stronger result: MSO_1-L model-checking with vertex labels L is not tractable for a graph class satisfying (a) and (b) of Theorem 2 unless *every* problem in the polynomial-time hierarchy is in $\mathrm{DTIME}(2^{o(n)})/\textsc{SubExp}$ (cf. Theorem 13).

Finally, as a corollary, we obtain an interesting consequence in the area of directed graph (digraph) width measures, improving upon [8]. Informally, digraph width measures that are subdigraph-monotone and algorithmically "powerful" is at most a poly-logarithmic factor of the tree-width of the underlying undirected graph—cf. Section 5. In this context, we let $U(D)$ denote the underlying undirected graph of a digraph D. Given a digraph width measure δ, we let $U_\delta(d) := \{U(D) \mid \delta(D) \leq d\}$ to be the set of underlying undirected graphs of digraphs of δ-width at most d.

▶ **Theorem 3** (reformulated as Theorem 15). *Assume a (suitable but fixed) finite label set L, and a digraph width measure δ such that*

a) δ is monotone under taking subdigraphs, and
b) $\mathrm{MC}(\mathrm{MSO}_1\text{-}L, \mathcal{D}^L)$, the MSO_1-L model-checking problem on all L-vertex-labeled digraphs $\mathcal{D}^L$ is in XP wrt. $\delta(D)$ and the input formula $\varphi \in \mathrm{MSO}_1$-$L$ as parameters.

Then, unless the non-uniform ETH fails, for all $d \in \mathbb{N}$ the tree-width of the class $U_\delta(d)$ is not densely unbounded poly-logarithmically.

Proof outline and organization

We are going to show via a suitable (multi-step) reduction that the potential tractability of MSO_1-L model-checking on our graph class implies sub-exponential time algorithms for problems which are not believed to have one (cf. ETH). The success of the reduction, of course, rests on the assumptions of $\mathcal{G}$ being subgraph-closed and of unbounded tree-width. So, at a high level, our proof technique is similar to that of Kreutzer and Tazari.

However, there are some crucial differences. While [13] uses the effectiveness assumption in Definition 1. ii and some further technically involved algorithms to construct a "skeleton" in the class $\mathcal{C}$ suitable for their reduction, in our reduction we obtain a corresponding labeled skeleton in the class $\mathcal{G}^L$ "for free" from an oracle advice function which comes with the non-uniform (fixed-sized circuits) computing model. That is why our complete proof is also significantly shorter than that in [13]. Additionally, our arguments employ a result on strong edge colorings of graphs in order to "simulate" certain edge sets within the MSO_1-L language, thus avoiding the need for a more expressive logic such as MSO_2.

The rest of the paper is organized as follows: In Section 2 we overview some standard terminology and notation. Section 3 then includes the core technical concepts: unbounding tree-width (Definition 8), the grid-like graphs of Reed and Wood [16] (Proposition 2), and a new way of interpreting arbitrary graphs in labeled grid-like graphs of sufficiently high order (Lemma 10). These then lead to the proof of our main result, equivalently formulated as Theorem 12, in Section 4. In this section, we also show the stronger collapse result in Theorem 13, that of $\mathrm{PH} \subseteq \mathrm{DTIME}(2^{o(n)})/\textsc{SubExp}$. The consequences for directed width measures are then discussed in Section 5, followed by concluding remarks in Section 6.

2 Preliminaries

The graphs we consider in this paper are *simple*, i.e. they do not contain loops and parallel edges. Given a graph G, we let $V(G)$ denote its vertex set and $E(G)$ its edge set. A *path P of length $r > 0$* in G is a sequence of vertices $P = (x_0, \ldots, x_r)$ such that all x_i are pairwise

distinct and $(x_i, x_{i+1}) \in E(G)$ for every $0 \leq i < r$. Let $\mathcal{S}$ be a family of sets S_i for $i = 1, 2, \ldots$. Then the *intersection graph on* $\mathcal{S}$ is the graph $I(\mathcal{S})$ where $V(I(\mathcal{S})) = \mathcal{S}$ and $S_iS_j \in E(I(\mathcal{S}))$ iff $S_i \cap S_j \neq \emptyset$.

Let $L = \{L_1, \ldots, L_k\}$ be a set of labels. A *L-vertex-labeled graph*, or *L-graph* for short, is a graph G together with a function $\lambda\colon V(G) \to 2^L$, assigning each vertex a set of labels, and we write (G, λ) to denote this graph. For a graph class $\mathcal{G}$, we shortly write $\mathcal{G}^L$ for the class of all L-graphs over $\mathcal{G}$, i.e. $\mathcal{G}^L$ contains all (G, λ) where $G \in \mathcal{G}$ and λ is an arbitrary L-vertex-labelling of G. Note that, unlike in e.g. [12], we do not allow labels for edges, which is in accordance with our focus on MSO$_1$ logic of graphs (defined next).

Monadic second-order logic (MSO) is an extension of first-order logic by quantification over sets. On the one-sorted adjacency model of graphs it reads as follows:

▶ **Definition 4.** The language of MSO$_1$, *monadic second-order logic of graphs*, contains the expressions built from the following elements:

i) variables $x, y, \ldots$ for vertices, and $X, Y, \ldots$ for sets of vertices,
ii) the predicates $x \in X$ and $adj(x, y)$ with the standard meaning,
iii) equality for variables, the connectives $\wedge, \vee, \neg, \to$ and the quantifiers $\forall, \exists$.

Note that we do not allow quantification over sets of edges (as edges are not elements). If we considered the two-sorted incidence graph model (in which the edges formed another sort of elements), we would obtain aforementioned MSO$_2$, *monadic second-order logic of graphs with edge-set quantification*, which is strictly more powerful than MSO$_1$, cf. [6]. Yet even MSO$_1$ has strong enough expressive power to describe many common problems.

▶ **Example 5.** The 3-Coloring problem can be expressed in MSO$_1$ as follows: $\exists V_1, V_2, V_3$ $\big[\forall v\,(v \in V_1 \vee v \in V_2 \vee v \in V_3) \wedge \bigwedge_{i=1,2,3} \forall v, w\,(v \notin V_i \vee w \notin V_i \vee \neg\, adj(v, w))\,\big]$.

The MSO$_1$ logic can naturally be extended to L-graphs. The *monadic second-order logic* on L-vertex-labeled graphs, denoted by MSO$_1$-L, is the natural extension of MSO$_1$ with unary predicates $L_i(x)$ for each label $L_i \in L$, such that $L_i(x)$ holds iff $L_i \in \lambda(x)$.

Parameterized complexity and MSO$_1$ model-checking

Throughout the paper we are interested in the problem of checking whether a given input graph satisfies a property specified by a fixed formula. This problem can be thought of as an instance of a parameterized problem, studied in the field of *parameterized complexity* (see e.g. [7] for a background on parameterized complexity).

A parameterized problem Q is a subset of $\Sigma \times \mathbb{N}_0$, where Σ is a finite alphabet and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$. A parameterized problem Q is said to be *fixed-parameter tractable* if there is an algorithm that given $(x, k) \in \Sigma \times \mathbb{N}_0$ decides whether (x, k) is a yes-instance of Q in time $f(k) \cdot p(|x|)$ where f is some computable function of k alone, p is a polynomial and $|x|$ is the size measure of the input. The class of such problems is denoted by FPT. The class XP is the class of parameterized problems that admit algorithms with a run-time of $O(|x|^{f(k)})$ for some computable f, i.e. polynomial-time for every fixed value of k.

We are dealing with a parameterized model-checking problem MC(MSO$_1, \mathcal{C}$) where $\mathcal{C}$ is a class of graphs; the task is to decide, given a graph $G \in \mathcal{C}$ and a formula $\phi \in$ MSO$_1$, whether $G \models \phi$. The parameter is $k = |\phi|$, the size of the formula ϕ. We actually consider the labeled variant MC(MSO$_1$-$L, \mathcal{C}$) for $\mathcal{C}$ being a class of L-graphs.

Interpretability of logic theories

One of our main tools is the classical interpretability of logic theories [15] (which in this setting is analogical to transductions as used e.g. by Courcelle [3]). To describe the simplified setting, assume that two classes of *relational structures* $\mathscr{K}$ and $\mathscr{L}$ are given. The basic idea of an *interpretation* I of the theory $\mathrm{Th}_{\mathrm{MSO}}(\mathscr{K})$ into $\mathrm{Th}_{\mathrm{MSO}}(\mathscr{L})$ is to transform MSO formulas ϕ over $\mathscr{K}$ into MSO formulas ϕ^I over $\mathscr{L}$ in such a way that "truth is preserved":

- First, one chooses a formula $\alpha(x)$ intended to define in each structure $G \in \mathscr{L}$ a set of individuals (new domain) $G[\alpha] := \{a : a \in dom(G) \text{ and } G \models \alpha(a)\}$, where $dom(G)$ denotes the set of individuals (domain) of G.
- Then, one chooses for each s-ary relational symbol R from $\mathscr{K}$ a formula $\beta^R(x_1, \ldots, x_s)$, with the intention to define a corresponding relation $G[\beta^R] := \{(a_1, \ldots, a_s) \colon a_1, \ldots, a_s \in dom(G) \text{ and } G \models \beta^R(a_1, \ldots, a_s)\}$. With these formulas one defines for each $G \in \mathscr{L}$ the relational structure $G^I := \big(G[\alpha], G[\beta^R], \ldots\big)$ intended to correspond with structures in $\mathscr{K}$.
- Finally, there is a natural way to translate each formula ϕ (over $\mathscr{K}$) into a formula ϕ^I (over $\mathscr{L}$), by induction on the structure of formulas. The atomic ones are substituted by corresponding chosen formulas (such as β^R) with the corresponding variables. Then one proceeds via induction simply as follows:

$$(\neg\phi)^I \;\mapsto\; \neg(\phi^I) \quad , \qquad (\phi_1 \wedge \phi_2)^I \;\mapsto\; (\phi_1)^I \wedge (\phi_2)^I,$$
$$\big(\exists x\, \phi(x)\big)^I \;\mapsto\; \exists y\, \big(\alpha(y) \wedge \phi^I(y)\big) \quad , \qquad \big(\exists X\, \phi(X)\big)^I \;\mapsto\; \exists Y\, \phi^I(Y).$$

The whole concept is shortly illustrated in by the following scheme

$$\begin{array}{ccc} \phi \in \text{MSO over } \mathscr{K} & \xrightarrow{\quad I \quad} & \phi^I \in \text{MSO over } \mathscr{L} \\ H \in \mathscr{K} & & G \in \mathscr{L} \\ G^I \cong H & \xleftarrow{\quad I \quad} & G \end{array}$$

▶ **Definition 6** (Interpretation between theories)**.** Let $\mathscr{K}$ and $\mathscr{L}$ be classes of relational structures. Theory $\mathrm{Th}_{\mathrm{MSO}}(\mathscr{K})$ is *interpretable* in theory $\mathrm{Th}_{\mathrm{MSO}}(\mathscr{L})$ if there exists an interpretation I as above such that the following two conditions are satisfied:

i) For every structure $H \in \mathscr{K}$, there is $G \in \mathscr{L}$ such that $G^I \cong H$, and
ii) for every $G \in \mathscr{L}$, the structure G^I is isomorphic to some structure of $\mathscr{K}$.

Furthermore, $\mathrm{Th}_{\mathrm{MSO}}(\mathscr{K})$ is *efficiently interpretable* in $\mathrm{Th}_{\mathrm{MSO}}(\mathscr{L})$ if the translation of each ϕ into ϕ^I is computable in polynomial time and the structure $G \in \mathscr{L}$, where $G^I \cong H$, can be computed from any $H \in \mathscr{K}$ in polynomial time.

Exponential-Time Hypothesis

The *Exponential-Time Hypothesis (ETH)*, formulated in [10], states that there exists no algorithm that can solve n-variable 3-SAT in time $2^{o(n)}$. It was shown in [10] that the hypothesis can be formulated using one of the many equivalent problems (e.g. k-Colorability or Vertex Cover)—i.e. sub-exponential complexity for one of these problems would imply the same for all the others.

ETH can be formulated in the *non-uniform* version: There is no family of algorithms (one for each input length) which can solve n-variable 3-SAT in time $2^{o(n)}$. In theory of computation literature, "non-uniform algorithms" are often referred to as "fixed-sized input circuits" where for each length of the input a different circuit is used. Yet another way of thinking about non-uniform algorithms is as having an algorithm that is allowed to receive an oracle advice, which depends only on the length of the input. As mentioned in [1], the results of [10] hold also for the non-uniform ETH.

3 Key Technical Concepts

Unbounding Tree-width

Following Definition 1, we aim to formally describe what it means to say that the tree-width of a graph class is not bounded by a function g. Recall (see also [12, 13]) that it is not enough just to assume $tw(G) > g(|V(G)|)$ for some sporadic values of tw with huge gaps between them, but a reasonable density of the surpassing tree-width values is also required. Hence we suggest the following alternative definition:

▶ **Definition 7** (Densely unbounded tree-width)**.** For a graph class $\mathcal{G}$, we say that the tree-width of $\mathcal{G}$ is *densely unbounded by a function* g if there is a constant $\gamma > 1$ such that, for every $m \in \mathbb{N}$, there exists a graph $G \in \mathcal{G}$ whose tree-width is $tw(G) \geq m$ and $|V(G)| < \mathcal{O}\big(g^{-1}(m^\gamma)\big)$. The constant γ is called the *gap-degree* of this property.

▶ Remark. Comparing to Definition 1 one can easily check that if the tree-width of a class $\mathcal{G}$ is strongly unbounded by a function g, then the tree-width is densely unbounded by g with the same gap-degree, and the witnessing graphs G of Definition 7 can be computed for all m efficiently—in sub-exponential time wrt. m. Hence our definition is weaker in this respect.

For simplicity we are interested in graph classes whose tree-width is densely unbounded by every poly-logarithmic function of the graph size. That is expressed by the following simpler definition:

▶ **Definition 8** (Densely unbounded tree-width II)**.** For a graph class $\mathcal{G}$, we say that the tree-width of $\mathcal{G}$ is *densely unbounded poly-logarithmically* if it is densely unbounded by $\log^c m$ for every $c \in \mathbb{N}$. That is, for every $c > 0$ the following holds: for all $m \in \mathbb{N}$ there exists a graph $G \in \mathcal{G}$ whose tree-width is $tw(G) \geq m$ and with size $|V(G)| < \mathcal{O}\big(2^{m^{1/c}}\big)$. (The gap-degree becomes irrelevant in this setting.)

Grid-like graphs

The notion of a grid-like minor was introduced by Reed and Wood in [16], and extensively used by Kreutzer and Tazari [14, 13]. In what follows, we avoid use of the word "minor" in our definition of the same concept, since "H-minors" where H is grid-like are always found as subgraphs of the target graph, which might cause some confusion.

▶ **Definition 9** (Grid-like [16])**.** A graph G together with a collection $\mathcal{P}$ of paths, formally the pair $(G, \mathcal{P})$, is called *grid-like* if the following is true:

i) G is the union of all the paths in $\mathcal{P}$,
ii) each path in $\mathcal{P}$ has at least two vertices, and
iii) the *intersection graph* $I(\mathcal{P})$ of the path collection is bipartite.

The *order* of such grid-like graph $(G, \mathcal{P})$ is the maximum integer ℓ such that the intersection graph $I(\mathcal{P})$ contains a K_ℓ-minor. When convenient, we refer to a grid-like graph simply as to G.

Note that the condition (ii) is not explicitly stated in [16], but its validity implicitly follows from the point to get a K_ℓ-minor in $I(\mathcal{P})$, cf. Theorem 2. One can easily observe the following:

▶ Proposition 1. Let $(G, \mathcal{P})$ be a grid-like graph. Then the collection $\mathcal{P}$ can be split into $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$ such that each $\mathcal{P}_i$, $i = 1, 2$, consists of pairwise disjoint paths. Consequently, the maximum degree in G is $\Delta(G) \leq 4$.

The next result is crucial for our paper:

▶ Proposition 2 (Reed and Wood [16]). Every graph with tree-width at least $c\ell^4\sqrt{\log \ell}$ contains a subgraph which is grid-like of order ℓ, for some constant c.

MSO_1 interpretation on grid-like graphs

Now we prove the core new technical tool of our paper. We show how the subgraphs of $I(\mathcal{P})$ of any grid-like graph $(G, \mathcal{P})$ can be efficiently MSO_1-interpreted in G itself with a suitable vertex labelling. First, we state a useful result about strong edge colorings of graphs—a *strong edge-coloring* is an assignment of colors to the edges of a graph such that no path of length three contains the same color twice.

▶ Proposition 3 (Cranston [5]). Every graph of maximum degree 4 has a strong edge-coloring using at most 22 colors. This coloring can be found with a polynomial-time algorithm.

For a class of grid-like graphs $\mathcal{G}$, let $I^{\subseteq}(\mathcal{G}) = \{H : H \subseteq I(\mathcal{P}),\ (G, \mathcal{P}) \in \mathcal{G}\}$ denote the class of all subgraphs of their intersection graphs. Our core tool is the following lemma.

▶ **Lemma 10.** *Let $\mathcal{G}$ be any class of grid-like graphs. There exists a fixed finite set L of labels, with $|L| \geq 47$, and a graph class $\mathcal{I} \supseteq I^{\subseteq}(\mathcal{G})$, such that the following holds. The* MSO_1 *theory of $\mathcal{I}$ has an efficient interpretation in the* MSO_1 *theory of $\mathcal{G}^L$—the class of all L-vertex-labeled graphs over $\mathcal{G}$. Stated differently, any $H \subseteq I(\mathcal{P})$ where $(G, \mathcal{P}) \in \mathcal{G}$ is interpreted in some L-graph of G.*

Proof. Note that the use of a class $\mathcal{I}$ in the statement of the lemma is only a technicality related to (ii) of Definition 6. We are actually interested only in interpreting the graphs from $I^{\subseteq}(\mathcal{G})$, and $\mathcal{I}$ then simply contains all the graphs that (also accidentally) result from the presented interpretation.

Hence we choose an arbitrary $(G, \mathcal{P}) \in \mathcal{G}$ and $H \subseteq I(\mathcal{P})$. The task is to find a vertex labeling $\lambda_H \colon V(G) \to 2^L$ such that H has an efficient MSO_1 interpretation in the labeled graph $(G, \lambda_H) \in \mathcal{G}^L$. By Proposition 3 (cf. also Proposition 1), let $\gamma \colon E(G) \to \{1, \ldots, 22\}$ be a strong edge-coloring of the chosen graph G. Let $\mathcal{P} = \mathcal{P}_w \cup \mathcal{P}_b$ be the bipartition of the paths forming G corresponding to the partite sets of $I(\mathcal{P})$. We call the paths of $\mathcal{P}_w \cap V(H)$ "white" and those of $\mathcal{P}_b \cap V(H)$ "black". The remaining paths not in the vertex set of H are irrelevant. The edges of white/black paths are also called white/black, respectively, with the understanding that some edges of G may be both white and black. For $x \in V(G)$, we let $w(x) = \{\gamma(f) : f \text{ is a white edge incident to } x\}$ and $b(x) = \{\gamma(f) : f \text{ is a black edge incident to } x\}$. According to Proposition 1, $|w(x)| \leq 2$, $|b(x)| \leq 2$.

The key observation, derived directly from the definition of a strong edge-coloring, is that any edge $f = xy \in E(G)$ is a white edge iff $w(x) \cap w(y) \neq \emptyset$, and analogously for black edges. This allows us to speak separately about the white and black edges in G using only the language of MSO_1. Another easy observation is that the vertex sets of the paths in $\mathcal{P}$ have a system of distinct representatives by Hall's theorem. For if $\mathcal{P}' \subseteq \mathcal{P}$ and $\mathcal{P}'$ contains p white paths and q black paths, then $|V(\mathcal{P}')| \geq 2 \cdot \max\{p, q\} \geq p + q$, proving Hall's criterion. We assign a marker $r(x) \in \{\emptyset, w, b\}$ to each $x \in V(G)$ such that $r^{-1}(w)$ is the set of the representatives of white paths and $r^{-1}(b)$ is that of black paths (i.e., $r^{-1}(\emptyset)$ are not representatives). Finally, we assign another vertex marker $m(x) \in \{0, 1\}$ to each vertex $x \in V(G)$ such that $m(x) = 1$ iff $x \in V(P_1) \cap V(P_2)$ where $P_1, P_2 \in V(H) \subseteq \mathcal{P}$ and $\{P_1, P_2\} \in E(H)$.

Hence the label set L consists of 22 "light" colors coming from γ values on white paths, another 22 "dark" colors from black paths, and the three singletons w, b, m described above

(altogether 47 binary labels). Note that the actual size of the needed label space over L is even much smaller; at most $\left[\binom{22}{2}+22+1\right]^2 \cdot 3 \cdot 2 < 2^{19}$. The label $\lambda_H(x)$ of a vertex $x \in V(G)$ then contains the disjoint union $w(x) \dot\cup b(x)$, the label $r(x)$ if $\neq \emptyset$, and finally m if $m(x) = 1$.

Now, the interpretation of H in (G, λ_H) is simply as follows: The domain, i.e. the vertex set of H, is identified within $V(G)$ by a predicate $\alpha(x)$ expressing that "$r(x) = w \vee r(x) = b$" in MSO$_1$-L. In formal logic language (cf. Section 2), it is $L_w(x) \vee L_b(x)$. The relational symbol adj of H is then replaced, for $x, y \in V(G)$ s.t. $\alpha(x) \wedge \alpha(y)$, with $\beta^{adj}(x,y) \equiv \exists z \big[$"$m(z) = 1$" $\wedge\, \varrho(x,z) \wedge \varrho(y,z)\big]$, where $\varrho(t,z) \equiv \big[$"$r(t) = w$" $\rightarrow con_w(t,z)\big] \wedge \big[$"$r(t) = b$" $\rightarrow con_b(t,z)\big]$ and where con_w (con_b) routinely expresses in MSO$_1$-L the fact that t, z belong to the same component induced by white (black) edges in G. Precisely, $con_w(t,z) \equiv \forall Z \big[\, z \in Z \wedge t \notin Z \rightarrow \exists u, v \big(v \in Z \wedge u \notin Z \wedge adj(u,v) \wedge$ "$w(u) \cap w(v) \neq \emptyset$"$\big)\big]$. Clearly, in this interpretation $(G, \lambda_H)^I \simeq H$ thanks to our choice of λ_H. ◀

Lemma 10 will be coupled with the next technical tool of similar flavor used in our previous [8]. We remark that its original formulation was even stronger, making the target graph class planar, but we are content with the following weaker formulation here. We call a graph G $\{1,3\}$-*regular* if all the vertices of G have degree either one or three.

▶ **Lemma 11** ([8, in Theorem 5.5]). *The* MSO$_1$ *theory of all simple graphs has an efficient interpretation in the* MSO$_1$ *theory of all simple* $\{1,3\}$*-regular graphs. Furthermore, this efficient interpretation I can be chosen such that, for every* MSO$_1$ *formula ψ, the resulting property ψ^I is invariant under subdivisions of edges; i.e. for every $\{1,3\}$-regular graph G and any subdivision G_1 of G it holds $G \models \psi^I$ iff $G_1 \models \psi^I$.*

4 The Main Theorem

▶ **Theorem 12** (cf. Theorem 2). *Let L be a finite set of labels, $|L| \geq 47$. Unless the nonuniform Exponential-Time Hypothesis fails, there exists no graph class $\mathcal{G}$ satisfying all the three properties*

a) $\mathcal{G}$ is closed under taking subgraphs,
b) the tree-width of $\mathcal{G}$ is densely unbounded poly-logarithmically,
c) the MC(MSO$_1$-$L, \mathcal{G}^L$) *model-checking problem is in* XP*, i.e. testing whether $G \models \varphi$ is solvable in time $\mathcal{O}\big(|V(G)|^{f(|\varphi|)}\big)$ for some computable function f.*

Proof. We will show that if there exists a graph class $\mathcal{G}$ satisfying all three properties stated above, then we contradict the non-uniform ETH. Fix $b \in \mathbb{N}$ (to be determined later) and any sufficiently large $c \in \mathbb{N}$ such that $c > 5b$. By (b) and Definition 8, we have that for all $m \in \mathbb{N}$ there is $G'_m \in \mathcal{G}$ such that $tw(G'_m) \geq m^{5b}$ and $|V(G'_m)| < \mathcal{O}\big(2^{m^{5b/c}}\big)$. By Proposition 2, the graph G'_m contains a subgraph $G_m \subseteq G'_m$ which is grid-like as $(G_m, \mathcal{P}_m)$ of order m^b, for all sufficiently large m. Also $G_m \in \mathcal{G}$ by (a). We fix (one of) the K_{m^b}-minor in $I(\mathcal{P}_m)$, and denote by $\mathcal{V}_m$ the partition of the vertex set of $I(\mathcal{P}_m)$ into connected subgraphs that define this minor. Furthermore, by Proposition 3, there exists a strong edge coloring $\gamma_m \colon E(G_m) \rightarrow \{1, \ldots, 22\}$ of G_m. Define an advice function A that acquires the values $A(m) := \langle G_m, \mathcal{P}_m, \mathcal{V}_m, \gamma_m \rangle$ (whenever m is large enough for G_m to be defined as above). Since $c > 5b$ and $|V(G_m)| < \mathcal{O}\big(2^{m^{5b/c}}\big)$, our advice function A is sub-exponentially bounded.

Now we get to the core of the proof. Assume that we get an arbitrary graph F and any MSO$_1$ formula φ as input. We show that the model-checking instance $F \models \varphi$ can be solved in sub-exponential time wrt. $m = |V(F)|$ with help of our advice function A. For starters we

query the oracle advice value $A(m) = \langle G_m, \mathcal{P}_m, \mathcal{V}_m, \gamma_m \rangle$. Then, by Lemma 11, there is an interpretation I_1 such that there exists a $\{1,3\}$-regular graph H and $H^{I_1} \simeq F$. Moreover, since I_1 is efficient, we can compute H efficiently and $|V(H)| \leq m^b$ for a suitable fixed b and sufficiently large m. Since our advice $(G_m, \mathcal{P}_m)$ is a grid-like graph of order m^b—i.e., its intersection graph $I(\mathcal{P}_m)$ has a K_{m^b}-minor—$I(\mathcal{P}_m)$ has a minor isomorphic to H, too. But H is $\{1,3\}$-regular and, in particular, has maximum degree three. Hence there exists a subgraph $H_1 \subseteq I(\mathcal{P}_m)$ that is isomorphic to a subdivision of H (in other words, H is a topological minor of $I(\mathcal{P}_m)$). This subgraph H_1 can be straightforwardly computed from the advice $\mathcal{V}_m$ over $(G_m, \mathcal{P}_m)$ in polynomial time.

By Lemma 10 there is another efficient interpretation I_2 assigning to H_1 a labeling λ_1 such that $(G_m, \lambda_1)^{I_2} \simeq H_1$. This λ_1 can actually be computed very easily with help of the advice γ_m from $A(m)$ along the lines of the proof of Lemma 10, not even using the algorithmic part of Proposition 3. Finally, we compute in polynomial time the formula $\psi \equiv (\varphi^{I_1})^{I_2}$. According to Lemma 11, ψ is invariant under subdivisions of edges, and so $H \models \varphi^{I_1} \iff H_1 \models \varphi^{I_1}$. Then, by the interpretation principle, $F \models \varphi \iff H \models \varphi^{I_1} \iff H_1 \models \varphi^{I_1} \iff (G_m, \lambda_1) \models \psi$. The final task is to run the algorithm of (c) on the instance $(G_m, \lambda_1) \models \psi$. The run-time is $|V(G_m)|^p$ for some p depending only on ψ, i.e. only on φ. Hence we get a solution to the model-checking instance $F \models \varphi$ in time $\mathcal{O}\big(|V(G_m)|^{f(|\varphi|)}\big) < \mathcal{O}\big(2^{f(|\varphi|)\cdot m^{5b/c}}\big) \in 2^{\mathcal{O}(m^{1-\varepsilon})}$ for any fixed φ, with a sub-exponentially bounded oracle advice function A.

In particular, if φ expresses the fact that a graph is 3-colorable (Example 5), then this shows that 3-Colorability $\in$ DTIME$(2^{o(m)})$/SubEXP, contradicting non-uniform ETH. ◀

▶ Proposition 4. Theorem 12 remains valid even if (b) is replaced with "the tree-width of $\mathcal{G}$ is densely unbounded by $\log^{q\cdot\gamma}$ with gap degree γ" for any $q > 8$.

Proof sketch. This follows from Definition 7 and since Lemma 11 works letting $b = 2$ (cf., [8]). Combining with Proposition 2, we see that any exponent $q > 2 \cdot 4$ suffices for our arguments to work, modulo the gap degree. ◀

We can strengthen Theorem 12 by showing that every problem in the Polynomial-Time Hierarchy (PH) is in DTIME$(2^{o(n)})$/SubEXP. But this stronger result comes at the price of a stricter assumption on the graph class $\mathcal{G}$: we assume that the MC(MSO$_1$-$L, \mathcal{G}^L$) model-checking problem is in XP (wrt. the formula size $|\varphi|$ as parameter) for *every* finite set of labels L such that $|L| = O(|\varphi|)$. Note that in Theorem 12, L was a fixed finite set of labels.

▶ **Theorem 13.** *Unless* PH $\subseteq$ DTIME$(2^{o(n)})$/SubEXP*, there exists no graph class $\mathcal{G}$ satisfying all three properties*

a) $\mathcal{G}$ is closed under taking subgraphs,
b) the tree-width of $\mathcal{G}$ is densely unbounded poly-logarithmically,
c) the MC(MSO$_1$-$L, \mathcal{G}^L$) *model-checking problem is in* XP*, i.e. testing whether $G \models \varphi$, where G is a vertex-labeled graph with $O(|\varphi|)$ labels, is solvable in time $\mathcal{O}\big(|V(G)|^{f(|\varphi|)}\big)$ for some computable function f.*

5 Implications for Directed Width Measures

In this section, we briefly foray into the area of digraph width measures and discuss the implications of the results in the previous section. This part follows on our earlier [8]. An important goal in the design of a "good" width measure is for it to satisfy two seemingly

contradictory requirements: (1) a large class of problems must be efficiently solvable on the graphs of bounded width; and, (2) the class of the graphs of bounded width should have a nice, reasonably rich and natural structure. In contrast to the undirected graph case, where e.g. tree-width has become a true success story, this effort has largely failed for digraph width measures. A partial answer for the reasons of this failure was provided in [8] where it was shown that any digraph width measure that is different from the undirected tree-width and monotone under directed topological minors is not algorithmically powerful. The phrase "different from tree-width" is defined by the property that there exists a constant $c \in \mathbb{N}$ such that the class of the underlying undirected graphs of digraphs of width at most c has unbounded tree-width. Algorithmic "powerfulness" has been defined as the property of admitting XP algorithms (wrt. the width as parameter) for all problems in MSO_1.

We improve upon this result by showing that even if the underlying undirected graphs corresponding to digraphs of bounded width have poly-logarithmically unbounded tree-width, and the digraph width measure is monotone just under subdigraphs, then the width measure is not algorithmically powerful. First note that we relax *unbounded* tree-width by *poly-logarithmically unbounded* tree-width. This is a somehow stronger assumption, and the strengthening is unavoidable due to a negative example shown in [8]. Secondly, we require the directed width measure to be closed under *subdigraphs* and not *directed topological minors* as in [8]; which is, on the other hand, a much weaker requirement. Thirdly, our interpretation of algorithmic powerfulness is that all problems in MSO_1-L can be solved on *L-vertex-labeled graphs* in XP-time wrt. the width and formula size as parameters. This again is a dilution of the notion of algorithmic power as defined in [8], where only plain MSO_1 over unlabeled digraphs has been exploited.

We start by defining what it means for a digraph width measure to have poly-logarithmically unbounded tree-width. We shortly denote by $U(D)$ the underlying undirected graph of a digraph D.

▶ **Definition 14.** A directed width measure δ *largely surpasses tree-width* if there exists $d \in \mathbb{N}$ such that the tree-width of the undirected graph class $\{U(D) : \delta(D) \leq d\}$ is densely unbounded poly-logarithmically.

Then the main result of this section reads:

▶ **Theorem 15.** *Let L be a finite set of labels, $|L| \geq 47$. Unless the non-uniform Exponential-Time Hypothesis fails, there exists no directed width measure δ satisfying all three properties:*

a) δ is monotone under taking subdigraphs;
b) δ largely surpasses the tree-width of underlying undirected graphs; and
c) for all L-vertex-labeled digraphs D and all formulas $\varphi \in MSO_1$-L, the problem of deciding whether $D \models \varphi$ is solvable in time $O(|D|^{f(\delta(D),|\varphi|)})$ for some computable f.

6 Concluding Remarks

Our paper contributes to Kreutzer and Tazari's impressive results in this area. Our proof is shorter and holds for MSO_1-L logic instead of MSO_2 at the price of a stronger assumption in computational complexity. The expressive power of MSO_2 over graphs with labels from a set L and MSO_1 with the same label set is huge—for instance, the latter is not able to express some natural graph problems like Hamiltonian cycle. However, one cannot directly compare the expressive power of bare MSO_2 without labels and MSO_1-L over graphs with vertex labels from L, as there are problems which can be expressed in MSO_1-L but not in

MSO_2 and vice versa. We have proved that it is not possible to efficiently process latter MSO_1-L on graph classes with "very" unbounded tree-width which are subgraph-closed.

Besides the implications discussed in Section 5, there is also an implication for another width measure—clique-width. Clique-width [4] (as well as rank-width) is a graph parameter which allows efficient (FPT time) model-checking of (labeled) MSO_1-L formulas, however it has received some criticism for not having nice structural properties such as being monotone under taking subgraphs. Our results indicate that it is unlikely any parameter exists with the desirable properties of clique-width which is monotone under taking subgraphs.

Finally, let us briefly mention the possibility of extending Theorem 12 to unlabeled graphs, i.e., using plain MSO_1 over $\mathcal{G}$ in Theorem 12 (c). It is not known whether there exists any natural and nontrivial graph class where unlabeled MSO_1 is efficiently solvable and yet MSO_1-L model-checking is hard. Such a graph class would necessarily contain graphs of unbounded clique-width (since otherwise MSO_1-L could be efficiently model-checked) and yet with sufficient structure to allow efficient model-checking of bare MSO_1.

References

1 V. Chandrasekaran, N. Srebro, and P. Harsha. Complexity of inference in graphical models. In *UAI'08*, pages 70–78, 2008.

2 B. Courcelle. The monadic second order logic of graphs I: Recognizable sets of finite graphs. *Inform. and Comput.*, 85:12–75, 1990.

3 B. Courcelle and J. Engelfriet. *Graph Structure and Monadic Second-Order Logic, a Language Theoretic Approach*. Cambridge University Press, April 2011. Book in preparation.

4 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.

5 D. W. Cranston. Strong edge-coloring of graphs with maximum degree 4 using 22 colors. *Discrete Math.*, 306(21):2772–2778, 2006.

6 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer, 1999.

7 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

8 R. Ganian, P. Hliněný, J. Kneis, D. Meister, J. Obdržálek, P. Rossmanith, and S. Sikdar. Are there any good digraph width measures? In *IPEC'10*, volume 6478 of *LNCS*, pages 135–146. Springer, 2010.

9 M. Grohe. Logic, graphs, and algorithms. In *Logic and Automata: History and Perspectives*, pages 357–422. Amsterdam University Press, 2008.

10 R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. System Sci.*, 63(4):512–530, 2001.

11 S. Kreutzer. Algorithmic meta-theorems. *Electronic Colloquium on Computational Complexity (ECCC)*, TR09-147, 2009.

12 S. Kreutzer. On the parameterised intractability of monadic second-order logic. In *CSL'09*, volume 5771 of *LNCS*, pages 348–363. Springer, 2009.

13 S. Kreutzer and S. Tazari. Lower bounds for the complexity of monadic second-order logic. In *LICS'10*, pages 189–198, 2010.

14 S. Kreutzer and S. Tazari. On brambles, grid-like minors, and parameterized intractability of monadic second-order logic. In *SODA'10*, pages 354–364, 2010.

15 M. O. Rabin. A simple method for undecidability proofs and some applications. In Y. Bar-Hillel, editor, *Logic, Methodology and Philosophy of Sciences*, volume 1, pages 58–68. North-Holland, Amsterdam, 1964.

16 B. Reed and D. Wood. Polynomial treewidth forces a large grid-like-minor. Technical Report abs/0809.0724, CoRR, 2008.

LP can be a cure for Parameterized Problems

N.S. Narayanaswamy[1], Venkatesh Raman[2], M.S. Ramanujan[2], and Saket Saurabh[2]

1 **Department of Computer Science and Engineering, IIT Madras, Chennai, India.**
swamy@cse.iitm.ernet.in

2 **The Institute of Mathematical Sciences, Chennai 600113, India.**
{vraman|msramanujan|saket}@imsc.res.in

Abstract

We investigate the parameterized complexity of Vertex Cover parameterized above the optimum value of the linear programming (LP) relaxation of the integer linear programming formulation of the problem. By carefully analyzing the change in the LP value in the branching steps, we argue that even the most straightforward branching algorithm (after some preprocessing) results in an $O^*(2.6181^r)$ algorithm for the problem where r is the excess of the vertex cover size over the LP optimum. We write $O^*(f(k))$ for a time complexity of the form $O(f(k)n^{O(1)})$, where $f(k)$ grows exponentially with k.

Then, using known and new reductions, we give $O^*(2.6181^k)$ algorithms for the parameterized versions of Above Guarantee Vertex Cover, Odd Cycle Transversal, Split Vertex Deletion and Almost 2-SAT, and an $O^*(1.6181^k)$ algorithm for Kŏnig Vertex Deletion, Vertex Cover Param by OCT and Vertex Cover Param by KVD. These algorithms significantly improve the best known bounds for these problems. The notable improvement is the bound for Odd Cycle Transversal for which this is the first major improvement after the first algorithm that showed it fixed-parameter tractable in 2003. We also observe that using our algorithm, one can obtain a simple kernel for the classical vertex cover problem with at most $2k - O(\log k)$ vertices.

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

Keywords and phrases Algorithms and data structures, Graph Algorithms, Parameterized Algorithms

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.338

1 Introduction and Motivation

In this paper we revisit one of the most studied problems in parameterized complexity, the Vertex Cover problem. Given a graph $G = (V, E)$, a subset $S \subseteq V$ is called *vertex cover* if every edge in E has at least one end-point in S. The Vertex Cover problem is formally defined as follows.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 338–349

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

Vertex Cover	
Instance:	An undirected graph G and a positive integer k.
Parameter:	k.
Problem:	Does G have a vertex cover of of size at most k?

We start with a few basic definitions regarding parameterized complexity. For decision problems with input size n, and a parameter k, the goal in parameterized complexity is to design an algorithm with runtime $f(k)n^{O(1)}$ where f is a function of k alone, as contrasted with a trivial $n^{f(k)}$ algorithm. Problems which admit such algorithms are said to be fixed parameter tractable (FPT). The theory of parameterized complexity was developed by Downey and Fellows [6]. For recent developments, see the book by Flum and Grohe [7].

Vertex Cover was one of the earliest problems that was shown to be FPT [6]. After a long race, the current best algorithm for Vertex Cover runs in time $O(1.2738^k + kn)$ [3]. However, when $k < m$, the size of the maximum matching, the Vertex Cover problem is not interesting, as the answer is trivially NO. And if m is large (suppose, for example, the graph has a perfect matching), then for the cases the problem is interesting, the running time of the standard version is not practical, as k, in this case, is quite large. This led to the following natural "above guarantee version" of the Vertex Cover problem.

Above Guarantee Vertex Cover (agvc)	
Instance:	An undirected graph G, a maximum matching M and a positive integer ℓ.
Parameter:	ℓ.
Problem:	Does G have a vertex cover of of size at most $\|M\| + \ell$?

The agvc problem is not only a very natural parameterization of the classical Vertex Cover problem but is also very central in the "zoo" of parameterized problems. We refer to the Figure 1 for the details of problems reducing to agvc. This implies that an improved algorithm for this problem implies improved algorithm for several other problems, including Almost 2-SAT and Odd Cycle Transversal.

The first known parameterized algorithm for agvc was using a parameter preserving reduction to Almost 2-SAT. In Almost 2-SAT, we are given a 2-SAT formula ϕ, a positive integer k and the objective is to check whether there exists at most k clauses whose deletion from ϕ can make the resulting formula satisfiable. The Almost 2-SAT problem was introduced in [16] and a decade later it was shown by Razgon and Barry O'Sullivan [23] to have an $O^*(15^k)$ time algorithm, thereby proving fixed-parameter tractability of the problem when k is the parameter. In 2011, there were two new algorithms for the agvc problem [5, 22]. One using new structural results about König-Egerváry graphs — graphs where the size of a minimum vertex cover is equal to the size of a maximum matching [22] and the other by a novel reduction to an "above guarantee version" of the Multiway Cut problem [5]. The second algorithm runs in time $O^*(4^k)$ and this is the previously fastest known algorithm for agvc.

The algorithm presented in [5] for agvc is not only the fastest known algorithm but also differs from previous algorithms conceptually in that it introduces the concept of approaching problems above the guarantee obtained by solving the relaxation of linear programming. This novel approach, combined with the fact that an improvement on the above guarantee

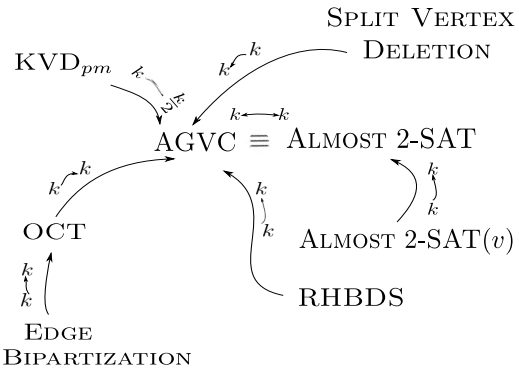


Figure 1 The zoo of problems around AGVC; An arrow from a problem P to a problem Q indicates that there is a parameterized reduction from P to Q with the parameter changes as indicated on the arrow.

versions of Vertex Cover improves the best known bounds for a number of parameterized problems, has motivated us to do a similar study for Vertex Cover.

The well known integer linear programming formulation (ILP) for Vertex Cover is as follows.

ILP formulation of Minimum Vertex Cover – ILPVC	
Instance:	A graph $G = (V, E)$.
Feasible Solution:	A function $x : V \to \{0, 1\}$ satisfying edge constraints $x(u) + x(v) \geq 1$ for each edge $(u, v) \in E$.
Goal:	To minimize $w(x) = \Sigma_{u \in V} x(u)$ over all feasible solutions x?

In the linear programming relaxation of the above ILP, the constraint $x(v) \in \{0, 1\}$ is replaced with $x(v) \geq 0$, for all $v \in V$. For a graph G, we call this relaxation LPVC(G). Clearly, every integer feasible solution is also a feasible solution to LPVC(G). If the minimum value of LPVC(G) is $vc^*(G)$ then clearly the size of a minimum vertex cover is at least $vc^*(G)$. So this leads to the following natural parameterization of Vertex Cover.

Vertex Cover above LP	
Instance:	An undirected graph G, positive integers k and $\lceil vc^*(G) \rceil$, where $vc^*(G)$ is the minimum value of LPVC(G)
Parameter:	$k - \lceil vc^*(G) \rceil$.
Problem:	Does G have a vertex cover of of size at most k?

Observe that since $vc^*(G) \geq m$, where m is the size of a maximum matching of G, we have that $k - vc^*(G) \leq k - m$. Thus any parameterized algorithm for Vertex Cover above LP is also an algorithm for AGVC and hence an algorithm for every problem described in Figure 1.

Our Results and Methodology. We develop an $O^*(2.6181^{(k-vc^*(G))})$ time for Vertex Cover above LP. Our algorithm is a simple branching algorithm. After a couple of preprocessing steps, the algorithm picks an arbitrary vertex v in the graph and recursively

Problem Name	Previous $f(k)$/Reference	New $f(k)$ in this paper
AGVC	4^k [5]	2.6181^k
Almost 2-SAT	4^k [5]	2.6181^k
RHorn-Backdoor Detection Set	4^k [5, 8]	2.6181^k
König Vertex Deletion	4^k [5, 18]	1.6181^k
Split Vertex Deletion	5^k [2]	2.6181^k
Odd Cycle Transversal	3^k [24]	2.6181^k
Vertex Cover Param by OCT	2^k (folklore)	1.6181^k
Vertex Cover Param by KVD	–	1.6181^k

Table 1 The table gives the previous $f(k)$ bound in the running time of various problems and the ones obtained in this paper.

tries to find a vertex cover of size at most k by considering whether v is in the solution or not. However, the analysis of the algorithm is more involved as it is not obvious that the measure $k - vc^*(G)$ will drop in the recursive steps. We string together several known results around linear programming relaxation of Vertex Cover to obtain this algorithm for Vertex Cover above LP. Some of the results we use are classical the Nemhauser-Trotter theorem and the properties of a "minimum surplus set". Using this algorithm we obtain an improved algorithm for every problem mentioned in Figure 1.

We give a list of problems with their previous best running time and the ones obtained in this paper in Table 1. The most notable one among them is the new algorithm for Odd Cycle Transversal, the problem of deleting at most k vertices to obtain a bipartite graph. The parameterized complexity of Odd Cycle Transversal was a long standing open problem in the area and only in 2003, Reed et al. [24], developed an algorithm for the problem running in time $O^*(3^k)$. In fact this was the first time that the iterative compression technique was used. However, there has been no further improvement over this algorithm in the last 9 years; though several reinterpretations of this algorithm have been published [9, 14].

We also find the algorithm for König Vertex Deletion, the problem of deleting at most k vertices to obtain a König graph very interesting. It is a natural generalization of the odd cycle transversal problem. In [18] it was shown that one can solve this problem by obtaining a minimum sized vertex cover of the graph and given a minimum vertex cover one can solve König Vertex Deletion in polynomial time. However, in this article we discover a relationship between the measure $k - vc^*(G)$ and the minimum number of vertices needed to delete to obtain a König graph, and this together with a reduction rule based on Nemhauser-Trotter theorem for König Vertex Deletion gives an algorithm with running time $O^*(1.6181^k)$.

We also note that using our algorithm, we obtain a simpler polynomial time algorithm for Vertex Cover that, given an input (G, k) returns an equivalent instance $(G' = (V', E'), k')$ such that $k' \leq k$ and $|V(G')| \leq 2k - c \log k$ for any fixed constant c. This is also known as a kernel for Vertex Cover in the literature. This improves the size bound on the previously known such algorithm [26], that gave an upper bound of $2k - c$ for any fixed constant c. Indepedently, Lampis [12] has also given a kernel for a Vertex Cover whose size is bounded by $2k - c \log k$.

We find this new algorithm for Odd Cycle Transversal and various other problems using an algorithm for Vertex Cover very exciting and hope that this will lead to a new race for Vertex Cover above LP like its classical counterpart Vertex Cover!

2 Preliminaries

Let $G = (V, E)$ denote a graph. For a subset S of V, the *subgraph of G induced by S* is denoted by $G[S]$ and it is defined as the subgraph of G with vertex set S and edge set $\{(u, v) \in E \ : \ u, v \in S\}$. By $N_G(u)$ we denote the (open) neighborhood of u, that is, the set of all vertices adjacent to u. Similarly, for a subset $T \subseteq V$, we define $N_G(T) = (\cup_{v \in T} N_G(v)) \setminus T$. When it is clear from the context, we drop the subscript G from the notation. The surplus of an independent set $X \subseteq V$ is defined as $\mathbf{surplus}(X) = |N(X)| - |X|$. The surplus of a graph G, $\mathbf{surplus}(G)$, is defined to be the minimum surplus over all independent sets in the graph.

By the phrase an *optimum solution* to LPVC(G), we mean a feasible solution with $x(v) \geq 0$ for all $v \in V$ minimizing the objective function $w(x) = \sum_{u \in V} x(u)$. It is well known that for any graph G, there exists an optimum solution to LPVC(G), such that $x(u) \in \{0, \frac{1}{2}, 1\}$ for all $u \in V$ [19]. Such a feasible optimum solution to LPVC(G) is called half integral and can be found in polynomial time [19]. In this paper we will always deal with half integral optimum solutions to LPVC(G). Thus by default whenever we will *say optimum solution* to LPVC(G) we will mean *half integral optimum solution* to LPVC(G). Let $VC(G)$ be the set of all minimum vertex covers of G and $vc(G)$ denote the size of a minimum vertex cover of G. Let $VC^*(G)$ be the set of all optimal solutions (including non half integral optimal solutions) to LPVC(G). By $vc^*(G)$ we denote the value of an optimum solution to LPVC(G). We define $V_i^x = \{u \in V : x(u) = i\}$ for each $i \in \{0, \frac{1}{2}, 1\}$ and define $x \equiv i$, $i \in \{0, \frac{1}{2}, 1\}$, if $x(u) = i$ for every $u \in V$. Clearly, $vc(G) \geq vc^*(G)$ and $vc^*(G) \leq \frac{|V|}{2}$ since $x = \frac{1}{2}$ is always a feasible solution to LPVC(G). We also refer to the $x \equiv \frac{1}{2}$ solution simply as the *all $\frac{1}{2}$ solution*. Proofs of results not appearing in the article will appear in the full version.

3 An Algorithm for Vertex Cover above LP

In this section we give an algorithm for Vertex Cover above LP. The algorithm has essentially two phases, a preprocessing phase and a branching phase. We first describe the preprocessing steps used in the algorithm and then give a simple description of the algorithm. Finally, we argue about its correctness and prove the desired running time bound on the algorithm.

3.1 Preprocessing

We describe two standard preprocessing rules to simplify the input instance. We first state the (known) results which allow for their correctness, and then describe the rules.

▶ **Lemma 1.** [20, 21] *For a graph G, in polynomial time, we can compute an optimal solution x to LPVC(G) such that all $\frac{1}{2}$ is the unique optimal solution to LPVC($G[V_{1/2}^x]$). Furthermore,* $\mathbf{surplus}(G[V_{1/2}^x]) > 0$.

▶ **Lemma 2.** [20] *Let G be a graph and x be an optimal solution to LPVC(G). There is a minimum vertex cover for G which contains all the vertices in V_1^x and none of the vertices in V_0^x.*

▶ Preprocessing Rule 1. Apply Lemma 1 to compute an optimal solution x to LPVC(G) such that all $\frac{1}{2}$ is the unique optimum solution to LPVC($G[V_{1/2}^x]$). If $V_0^x \cup V_1^x \neq \emptyset$ then delete the vertices in $V_0^x \cup V_1^x$ from the graph and reduce k by $|V_1^x|$.

The soundness/correctness of Preprocessing Rule 1 follows from Lemma 2. After the application of preprocessing rule 1, we know that $x \equiv \frac{1}{2}$ is the unique optimal solution to LPVC() of the resulting graph and the graph has a surplus of at least 1. This brings us to the next lemma which allows us to compute an independent set of a minimum surplus.

▶ **Lemma 3.** (see Theorem 6.1.4 in [15], see also [4] and [20]) *Given a graph G, the surplus of G, i.e. an independent set in G of minimum surplus, can be computed in polynomial time.*

▶ **Lemma 4.** [3, 20] *Let G be a graph, and let $Z \subseteq V(G)$ be an independent set such that* $\mathbf{surplus}(Z) = 1$ *and for every $Y \subseteq Z$,* $\mathbf{surplus}(Y) \geq \mathbf{surplus}(Z)$. *Then,*

1. *If the graph induced by $N(Z)$ is not an independent set, then there exists a minimum vertex cover in G that includes all of $N(Z)$ and excludes all of Z.*
2. *If the graph induced by $N(Z)$ is an independent set, let G' be the graph obtained from G by removing $Z \cup N(Z)$ and adding a vertex z, followed by making z adjacent to every vertex $v \in G \setminus (Z \cup N(Z))$ which was adjacent to a vertex in $N(Z)$ (also called* identifying *the vertices of $N(Z)$). Then, G has a vertex cover of size at most k if and only if G' has a vertex cover of size at most $k - |Z|$.*

▶ Preprocessing Rule 2. Using Lemma 3, find the minimum surplus independent set Z in G. If $\mathbf{surplus}(Z) = 1$, then apply Lemma 4 to reduce the instance. In other words, if the graph induced by $N(Z)$ is not an independent set, then include $N(Z)$ in the vertex cover, delete $Z \cup N(Z)$ from the graph, and decrease k by $|N(Z)|$ and otherwise, remove Z from the graph, identify the vertices of $N(Z)$, and decrease k by $|Z|$.

The soundness of Preprocessing Rule 2 follows from Lemma 4. As Z is a minimum surplus set in G, for every $Y \subseteq Z$, $\mathbf{surplus}\ (Y) \geq \mathbf{surplus}(Z)$.

After the exhaustive application of Preprocessing rules 1 and 2, for the resulting graph, all $\frac{1}{2}$ is the unique optimum solution to the LPVC() and the graph has a surplus of at least 2.

3.2 Branching

After the preprocessing rules are applied exhaustively until neither of the rules apply, we pick an arbitrary vertex u in the graph and branch on it. In other words, in one branch, we add u into the vertex cover, decrease k by 1, and delete u from the graph, and in the other branch, we add $N(u)$ into the vertex cover, decrease k by $|N(u)|$, and delete $\{u\} \cup N(u)$ from the graph. The correctness of this algorithm follows from the soundness of the preprocessing rules and the fact that the branching is exhaustive.

3.3 Analysis

In order to analyze the running time of our algorithm, we define a measure $\mu = \mu(G, k) = k - vc^*(G)$. We will first show that our preprocessing rules do not increase this measure. Following this, we will prove a lower bound on the decrease in the measure occurring as a result of the branching, thus allowing us to bound the running time of the algorithm in terms of the measure μ. For each case, we let (G', k') be the instance resulting by the application of the rule or branch, and let x' be an optimum solution to LPVC(G').

1. Consider the application of Preprocessing Rule 1. We know that $k' = k - |V_1^x|$. Since $x' \equiv \frac{1}{2}$ is the unique optimum solution to LPVC(G'), and G' comprises precisely the

vertices of $V^x_{1/2}$, the value of the optimum solution to LPVC(G') is exactly $|V^x_1|$ less than that of G. Hence, $\mu(G,k) = \mu(G',k')$.

2. We now consider the application of Preprocessing Rule 2.

(a) Suppose that $N(Z)$ was not independent. In this case, $k' = k - |N(Z)|$. We also know that $w(x') = \sum_{u \in V} x'(u) = w(x) - \frac{1}{2}(|Z| + |N(Z)|) + \frac{1}{2}(|V_1^{x'}| - |V_0^{x'}|)$. Adding and subtracting $\frac{1}{2}(|N(Z)|)$, we get $w(x') = w(x) - |N(Z)| - \frac{1}{2}(|Z| - |N(Z)|) + \frac{1}{2}(|V_1^{x'}| - |V_0^{x'}|)$. But, $Z \cup V_0^{x'}$ is an independent set in G, and $N(Z \cup V_0^{x'}) = N(Z) \cup V_1^{x'}$ in G. Since $\mathbf{surplus}(G) \geq 1$, $|N(Z \cup V_0^{x'})| - |Z \cup V_0^{x'}| \geq 1$. Hence, $w(x') = w(x) - |N(Z)| + \frac{1}{2}(|N(Z \cup V_0^{x'})| - |Z \cup V_0^{x'}|) \geq w(x) - |N(Z)| + \frac{1}{2}$. Thus, $\mu(G',k') \leq \mu(G,k) - \frac{1}{2}$.

(b) Suppose that $N(Z)$ was independent. In this case, $k' = k - |Z|$. We claim that $w(x') \geq w(x) - |Z|$. Suppose that this is not true. Then, it must be the case that $w(x') \leq w(x) - |Z| - \frac{1}{2}$. We will now consider three cases depending on the value $x'(z)$ where z is the vertex in G' resulting from the identification of $N(Z)$.

Case 1: $x'(z) = 1$. Now consider the following function $x'' : V \to \{0, \frac{1}{2}, 1\}$. For every vertex v in $G' \setminus \{z\}$, retain the value assigned by x', that is $x''(v) = x'(v)$. For every vertex in $N(Z)$, assign 1 and for every vertex in Z, assign 0. Clearly this is a feasible solution. But now, $w(x'') = w(x') - 1 + |N(Z)| = w(x') - 1 + (|Z| + 1) \leq w(x) - \frac{1}{2}$. Hence, we have a feasible solution of value less than the optimum, which is a contradiction.

Case 2: $x'(z) = 0$. Now consider the following function $x'' : V \to \{0, \frac{1}{2}, 1\}$. For every vertex v in $G' \setminus \{z\}$, retain the value assigned by x', that is $x''(v) = x'(v)$. For every vertex in Z, assign 1 and for every vertex in $N(Z)$, assign 0. Clearly this is a feasible solution. But now, $w(x'') = w(x') + |Z| \leq w(x) - \frac{1}{2}$. Hence, we have a feasible solution of value less than the optimum, which is a contradiction.

Case 3: $x'(z) = \frac{1}{2}$. Now consider the following function $x'' : V \to \{0, \frac{1}{2}, 1\}$. For every vertex v in $G' \setminus \{z\}$, retain the value assigned by x', that is $x''(v) = x'(v)$. For every vertex in $Z \cup N(Z)$, assign $\frac{1}{2}$. Clearly this is a feasible solution. But now, $w(x'') = w(x') - \frac{1}{2} + \frac{1}{2}(2|Z|) + \frac{1}{2} \leq w(x) - \frac{1}{2}$. Hence, we have a feasible solution of value less than the optimum, which is a contradiction.

Hence, $w(x') \geq w(x) - |Z|$, which implies that $\mu(G',k') \leq \mu(G,k)$.

3. We now consider the branching step.

a. Consider the case when we pick u in the vertex cover. In this case, $k' = k - 1$. We claim that $w(x') \geq w(x) - \frac{1}{2}$. Suppose that this is not the case. Then, it must be the case that $w(x') \leq w(x) - 1$. Consider the following assignment $x'' : V \to \{0, \frac{1}{2}, 1\}$ to LPVC(G). For every vertex $v \in V \setminus \{u\}$, set $x''(v) = x'(v)$ and set $x''(u) = 1$. Now, x'' is clearly a feasible solution and has a value at most that of x. But this contradicts our assumption that $x \equiv \frac{1}{2}$ is the unique optimum solution to LPVC(G). Hence, $w(x') \geq w(x) - \frac{1}{2}$, which implies that $\mu(G',k') \leq \mu(G,k) - \frac{1}{2}$.

b. Consider the case when we don't pick u in the vertex cover. In this case, $k' = k - |N(u)|$. We know that $w(x') = w(x) - \frac{1}{2}(|\{u\}| + |N(u)|) + \frac{1}{2}(|V_1^{x'}| - |V_0^{x'}|)$. Adding and subtracting $\frac{1}{2}(|N(u)|)$, we get $w(x') = w(x) - |N(u)| - \frac{1}{2}(|\{u\}| - |N(u)|) + \frac{1}{2}(|V_1^{x'}| - |V_0^{x'}|)$. But, $\{u\} \cup V_0^{x'}$ is an independent set in G, and $N(\{u\} \cup V_0^{x'}) = N(u) \cup V_1^{x'}$ in G. Since $\mathbf{surplus}(G) \geq 2$, $|N(\{u\} \cup V_0^{x'})| - |\{u\} \cup V_0^{x'}| \geq 2$. Hence, $w(x') = w(x) - |N(u)| + \frac{1}{2}(|N(\{u\} \cup V_0^{x'})| - |\{u\} \cup V_0^{x'}|) \geq w(x) - |N(u)| + 1$.

Hence, $\mu(G',k') \leq \mu(G,k) - 1$.

We have thus shown that the preprocessing rules do not increase the measure $\mu(G,k)$ and the branching step results in a $(\frac{1}{2},1)$ decrease in $\mu(G,k)=\mu$, resulting in the recurrence $T(\mu) \leq T(\mu-\frac{1}{2})+T(\mu-1)$ which solves to $(2.6181)^{\mu}=(2.6181)^{k-vc^*(G)}$. Thus we get a $(2.6181)^{(k-vc^*(G))}$ algorithm for Vertex Cover above LP.

▶ **Theorem 5.** Vertex Cover above LP *can be solved in time* $O^*((2.6181)^{k-vc^*(G)})$.

By applying the above theorem iteratively for increasing values of k, we can compute a minimum vertex cover of G and hence we have the following corollary.

▶ **Corollary 6.** *There is an algorithm that, given a graph G, computes a minimum vertex cover of G in time* $O^*(2.6181^{(vc(G)-vc^*(G))})$.

4 Applications

In this section we give several applications of the algorithm developed for Vertex Cover above LP.

4.1 An algorithm for Above Guarantee Vertex Cover

Since the value of the LP relaxation is at least the size of the maximum matching, our algorithm also runs in time $O^*(2.6181^{k-m})$ where k is the size of the minimum vertex cover and m is the size of the maximum matching.

▶ **Theorem 7.** Above Guarantee Vertex Cover *can be solved in time* $O^*(2.6181^{\ell})$ *time, where ℓ is the excess of the minimum vertex cover size above the size of the maximum matching.*

Now by the known reductions in [8, 17, 22] (see also Figure 1) we get the following corollary to Theorem 7.

▶ **Corollary 8.** Almost 2-SAT, Almost 2-SAT(v), RHorn-Backdoor Detection Set *can be solved in time* $O^*(2.6181^k)$. *However,* KVD_{pm} *can be solved in time* $O^*(1.6181^k)$.

4.2 Algorithms for Odd Cycle Transversal and Split Vertex Deletion

We describe a generic algorithm for both Odd Cycle Transversal and Split Vertex Deletion. Let $X, Y \in \{\text{Clique, Independent Set}\}$. A graph G is called an (X,Y)-*graph* if its vertices can be partitioned into X and Y. Observe that when X and Y are both *independent set*, this corresponds to a *bipartite graph* and when X is *clique* and Y is *independent set*, this corresponds to a *split graph*. In this section we outline an algorithm that runs in time $O^*(2.6181^k)$ and solves the following problem.

(X,Y)-Transversal Set

Instance:	An undirected graph G and a positive integer k.
Parameter:	k.
Problem:	Does G have a vertex subset S of size at most k such that its deletion leaves a (X,Y)-graph?

We solve the (X,Y)-Transversal Set problem by using a reduction to AGVC that takes k to k [25].

▶ **Theorem 9.** (X,Y)-Transversal Set *can be solved in time* $O^*(2.6181^k)$.

As a corollary to the above theorem we get the following new results.

▶ **Corollary 10.** Odd Cycle Transversal *and* Split Vertex Deletion *can be solved in time* $O^*(2.6181^k)$.

4.3 An algorithm for König Vertex Deletion

A graph G is called König if the size of a minimum vertex cover equals that of a maximum matching in the graph. Clearly bipartite graphs are König but there are non-bipartite graphs that are König (a triangle with an edge attached to one of its vertices, for example). The König Vertex Deletion problem is stated as follows.

König Vertex Deletion (KVD)	
Instance:	An undirected graph G and a positive integer k.
Parameter:	k.
Problem:	Does G have a vertex subset S of size at most k such that $G \setminus S$ is a König graph?

This problem is a natural generalization of the Odd Cycle Transversal problem. If the input graph G to König Vertex Deletion has a perfect matching then this problem is called KVD_{pm}. By Corollary 8, we already know that KVD_{pm} has an algorithm with running time $O^*(1.6181^k)$ by a polynomial time reduction to AGVC, that takes k to $k/2$. However, there is no known reduction if we do not assume that the input graph has a perfect matching and it required several interesting structural theorems in [18] to show that KVD can be solved as fast as AGVC. Here, we outline an algorithm for KVD that runs in $O^*(1.6181^k)$ and uses an interesting reduction rule. However, for our algorithm we take a slight detour and solve a slightly different, although equally interesting problem. Given a graph, a set S of vertices is called *König vertex deletion set (kvd set)* if its removal leaves a König graph. The auxiliary problem we study is following.

Vertex Cover Param by KVD	
Instance:	An undirected graph G, a König vertex deletion set S of size at most k and a positive integer ℓ.
Parameter:	k.
Problem:	Does G have a vertex cover of size at most ℓ?

This fits into the recent study of problems parameterized by other structural parameters. See, for example Odd Cycle Transversal parameterized by various structural parameters [11] or Treewidth parameterized by vertex cover [1] or Vertex Cover parameterized by feedback vertex set [10].

For our proofs we will use the following characterization of König graphs.

▶ **Lemma 11.** [18, Lemma 1] *A graph* $G = (V, E)$ *is König if and only if there exists a bipartition of* V *into* $V_1 \uplus V_2$*, with* V_1 *a vertex cover of* G *such that there exists a matching across the cut* (V_1, V_2) *saturating every vertex of* V_1.

Note that in Vertex Cover param by KVD, $G \setminus S$ is a König graph. So one could branch on all subsets of S to include in the output vertex cover, and for those elements not picked in S, we could pick its neighbors in $G \setminus S$ and delete them. However, the resulting graph need not be König adding to the complications. Note, however, that such an algorithm would yield an $O^*(2^k)$ algorithm for Vertex Cover Param by OCT. That is, if S were an odd cycle transversal then the resulting graph after deleting the neighbors of vertices not picked from S will remain a bipartite graph, where an optimum vertex cover can be found in polynomial time.

Given a graph $G = (V, E)$ and two disjoint vertex subsets V_1, V_2 of V, we let (V_1, V_2) denote the bipartite graph with vertex set $V_1 \cup V_2$ and edge set $\{\{u, v\} : \{u, v\} \in E \text{ and } u \in V_1, v \in V_2\}$. Now, we describe an algorithm based on Theorem 5, that solves Vertex Cover param by KVD in time $O^*(1.6181^k)$.

▶ **Theorem 12.** Vertex Cover Param by KVD *can be solved in time* $O^*(1.6181^k)$.

Proof. Let G be the input graph, S be a kvd set of size at most k. We first apply Lemma 1 on $G = (V, E)$ and obtain an optimum solution to LPVC(G) such that all $\frac{1}{2}$ is the unique optimum solution to LPVC($G[V^x_{1/2}]$). Due to Lemma 2, this implies that there exists a minimum vertex cover of G that contains all the vertices in V^x_1 and none of the vertices in V^x_0. Hence, the problem reduces to finding a vertex cover of size $\ell' = \ell - |H|$ for the graph $G' = G[V^x_{1/2}]$. Before we describe the rest of the algorithm, we prove the following lemma regarding kvd sets in G and G' which shows that if G has a kvd set of size at most k then so does G'. Even though this looks straight forward, the fact that König graphs are not hereditary (i.e. induced subgraphs of König graphs need not be König) makes this a non-trivial claim to prove.

▶ **Lemma 13.** *Let G and G' be defined as above. Let S be a kvd set of graph G of size at most k. Then, there is a kvd set of graph G' of size at most k.*

We now show that $\mu = vc(G') - vc^*(G') \leq \frac{k}{2}$. Let O be a kvd set of G' and define G'' as the Kónig graph $G' \setminus O$. We know that $|M| = vc(G'') = vc^*(G'')$, where M is a maximum matching in the graph G''. This implies that $vc(G') \leq vc(G'') + |O| = |M| + |O|$. But, we also know that $vc^*(G') \geq |M| + \frac{1}{2}(|O|)$ and hence, $vc(G') - vc^*(G') \leq \frac{1}{2}(|O|)$. By Lemma 13, we know that there is an O such that $|O| \leq k$ and hence, $vc(G') - vc^*(G') \leq \frac{k}{2}$.

By Corollary 6, we can find a minimum vertex cover of G' in time $O^*(2.6181^{vc(G')-vc^*(G')})$ and hence in time $O^*(2.6181^{k/2})$. If the size of the minimum vertex cover obtained for G' is at most ℓ', then we return yes else we return no. This completes the proof of the theorem. ◀

It is known that, given a minimum vertex cover, a minimum sized kvd set can be computed in polynomial time [18]. Hence, Theorem 12 has the following corollary.

▶ **Corollary 14.** KVD *can be solved in time* $O^*(1.6181^k)$.

Since the size of a minimum Odd Cycle Transversal is at least the size of a minimum Konig Vertex Deletion set, we also have the following corollary.

▶ **Corollary 15.** Vertex Cover Param by OCT *can be solved in time* $O^*(1.6181^k)$.

4.4 An improved kernel for Vertex Cover

We give a kernelization for Vertex Cover based on Theorem 5 as follows. Exhaustively, apply the Preprocessing rules 1 and 2 (see Section 3). When the rules no longer apply, if $k - vc^*(G) \leq \log k$, then solve the problem in time $O^*(2.6181^{\log k}) = O(n^{O(1)})$. Otherwise, just return the instance. We claim that the number of vertices in the returned instance is at most $2k - 2\log k$. Since $k - vc^*(G) > \log k$, $vc^*(G)$ is upper bounded by $k - \log k$. But, we also know that when Preprocessing Rule 1 is no longer applicable, all $\frac{1}{2}$ is the unique optimum to LPVC(G) and hence, the number of vertices in the graph G is twice the value of the optimum value of LPVC(G). Hence, $|V| = 2vc^*(G) \leq 2(k - \log k)$. Observe that by the same method we can also show that in the reduced instance the number of vertices is upper bounded by $2k - c\log k$ for any fixed constant c. Indepedently, Lampis [12] has also given a kernel for a Vertex Cover whose size is bounded by $2k - c\log k$.

5 Conclusion and Further Work

We have demonstrated that using the drop in LP values to analyze branching algorithms can give powerful results for parameterized complexity. Recently, in [13], a significantly faster algorithm for Vertex Cover above LP, running in time $O^*(2.3146^k)$, has been obtained. We believe that our algorithm is the beginning of a race to improve the running time bound for agvc and possibly for the classical vertex cover problem, for which there has been no progress in the last several years after an initial plethora of results.

Our other contribution is to exhibit several parameterized problems that are equivalent to or reduce to agvc through parameterized reductions. We observe that as the parameter change in these reductions are linear, any upper or lower bound results for kernels for one problem will carry over for the other problems too (subject to the directions of the reductions).

References

1 H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Preprocessing for treewidth: A combinatorial analysis through kernelization. In L. Aceto, M. Henzinger, and J. Sgall, editors, *ICALP (1)*, volume 6755 of *Lecture Notes in Computer Science*, pages 437–448. Springer, 2011.

2 L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Inf. Process. Lett.*, 58(4):171–176, 1996.

3 J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40-42):3736–3756, 2010.

4 M. Chlebík and J. Chlebíková. Crown reductions for the minimum weighted vertex cover problem. *Discrete Applied Mathematics*, 156(3):292–312, 2008.

5 M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. O. Wojtaszczyk. On multiway cut parameterized above lower bounds. In *IPEC*, 2011.

6 R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer-Verlag, New York, 1999.

7 J. Flum and M. Grohe. *Parameterized Complexity Theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

8 G. Gottlob and S. Szeider. Fixed-parameter algorithms for artificial intelligence, constraint satisfaction and database problems. *Comput. J.*, 51(3):303–325, 2008.

9 F. Hüffner. Algorithm engineering for optimal graph bipartization. *J. Graph Algorithms Appl.*, 13(2):77–98, 2009.

10 B. M. P. Jansen and H. L. Bodlaender. Vertex cover kernelization revisited: Upper and lower bounds for a refined parameter. In T. Schwentick and C. Dürr, editors, *STACS*, volume 9 of *LIPIcs*, pages 177–188. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2011.

11 B. M. P. Jansen and S. Kratsch. On polynomial kernels for structural parameterizations of odd cycle transversal. *CoRR*, abs/1107.3658, To appear in IPEC 2011.

12 M. Lampis. A kernel of order $2k - c \log k$ for vertex cover. *Inf. Process. Lett.*, 111 (23-24):1089–1091, 2011.

13 D. Lokshtanov, N. S. Narayanaswamy, V. Raman, M. S. Ramanujan, and S. Saurabh. Improved Parameterized Algorithms using LP. *Manuscript*, 2012.

14 D. Lokshtanov, S. Saurabh, and S. Sikdar. Simpler parameterized algorithm for oct. In J. Fiala, J. Kratochvíl, and M. Miller, editors, *IWOCA*, volume 5874 of *Lecture Notes in Computer Science*, pages 380–384. Springer, 2009.

15 L. Lovász and M. D. Plummer. *Matching Theory*. North Holland, Oxford, 1986.

16 M. Mahajan and V. Raman. Parameterizing above guaranteed values: Maxsat and maxcut. *J. Algorithms*, 31(2):335–354, 1999.

17 D. Marx and I. Razgon. Constant ratio fixed-parameter approximation of the edge multicut problem. *Inf. Process. Lett.*, 109(20):1161–1166, 2009.

18 S. Mishra, V. Raman, S. Saurabh, S. Sikdar, and C. R. Subramanian. The complexity of könig subgraph problems and above-guarantee vertex cover. *Algorithmica*, 58, 2010.

19 G. L. Nemhauser and L. E. Trotter. Properties of vertex packing and independence system polyhedra. *Mathematical Programming*, 6:48–61, 1974. 10.1007/BF01580222.

20 G. L. Nemhauser and L. E. Trotter. Vertex packings: Structural properties and algorithms. *Mathematical Programming*, 8:232–248, 1975. 10.1007/BF01580444.

21 J.-C. Picard and M. Queyranne. On the integer-valued variables in the linear vertex packing problem. *Mathematical Programming*, 12(1):97–101, 1977.

22 V. Raman, M. S. Ramanujan, and S. Saurabh. Paths, flowers and vertex cover. In C. Demetrescu and M. Halldórsson, editors, *Algorithms – ESA 2011*, volume 6942 of *Lecture Notes in Computer Science*, pages 382–393. Springer Berlin / Heidelberg, 2011.

23 I. Razgon and B. O'Sullivan. Almost 2-sat is fixed-parameter tractable. *J. Comput. Syst. Sci.*, 75(8):435–450, 2009.

24 B. A. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Oper. Res. Lett.*, 32(4):299–301, 2004.

25 S. Saurabh. *Exact Algorithms for some parameterized and optimization problems on graphs*. PhD thesis, 2008.

26 A. Soleimanfallah and A. Yeo. A kernel of order $2k - c$ for vertex cover. *Discrete Mathematics*, 311(10-11):892–895, 2011.

Mind Change Speed-up for Learning Languages from Positive Data *

Sanjay Jain[1] and Efim Kinber[2]

1 **School of Computing, National University of Singapore, Singapore 117417, Republic of Singapore.** sanjay@comp.nus.edu.sg
2 **Department of Computer Science, Sacred Heart University, Fairfield, CT 06825-1000, U.S.A.** kinbere@sacredheart.edu

Abstract

Within the frameworks of learning in the limit of indexed classes of recursive languages from positive data and automatic learning in the limit of indexed classes of regular languages (with automatically computable sets of indices), we study the problem of minimizing the maximum number of mind changes $\mathbf{F}_{\mathbf{M}}(n)$ by a learner $\mathbf{M}$ on all languages with indices not exceeding n. For inductive inference of recursive languages, we establish two conditions under which $\mathbf{F}_{\mathbf{M}}(n)$ can be made smaller than any recursive unbounded non-decreasing function. We also establish how $\mathbf{F}_{\mathbf{M}}(n)$ is affected if at least one of these two conditions does not hold. In the case of automatic learning, some partial results addressing speeding up the function $\mathbf{F}_{\mathbf{M}}(n)$ are obtained.

1998 ACM Subject Classification F.0 Theory of Computation ,I.2.6 Learning

Keywords and phrases Algorithmic and automatic learning, mind changes, speedup.

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.350

1 Introduction

In this paper, we consider a popular model for learning languages in the limit from infinite positive data (inductive inference), as defined by M. Gold in [13] (in the sequel, we refer to it as **TxtEx**): a learner is an algorithmic device that, given access to potentially all positive data (as a stream of data items, intermittent with a special character representing "no data at this moment"), produces a (potentially infinite) sequence of conjectures, and eventually stabilizes on a correct grammar for the target language. Specifically, we concentrate on learnability of *indexed* classes of languages — represented by computable numberings of languages with uniformly decidable membership problem; these classes represent practically interesting families of languages, in particular, the class of regular languages as represented by all finite automata or regular expressions and its practically important subclasses, and the class of pattern languages represented by patterns [1].

There are many different measures of complexity for learning languages in the limit [8, 9, 10, 22, 16, 12]. One obvious natural measure of complexity is the number of mind changes that a learner makes on a target language before stabilizing on a correct grammar for it. As there are infinitely many languages in the target class, it is natural to consider the maximum number of mind changes that a learner $\mathbf{M}$ makes on the first $n+1$ languages in the numbering defining the target class; in the sequel, we denote this number by $\mathbf{F}_{\mathbf{M}}(n)$ (another approach to mind change complexity was suggested in [19]). This measure of complexity of

* Sanjay Jain was supported in part by NUS grant number C-252-000-087-001.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 350–361

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


inductive inference, in the context of learning indexed families of recursive functions, was first suggested by J. Bārzdiņš and R. Freivalds in [4], where they also initiated a study of the bounds on the function $\mathbf{F}_{\mathbf{M}}(n)$. It is easy to see that $\mathbf{F}_{\mathbf{M}}(n)$ can be bounded by n — the learner can use the "identification by enumeration" strategy, whereby all functions in the numbering consistent with the input data seen so far are tried, starting from the first one, until a (smallest) index of the target function is found. However, Bārzdiņš and Freivalds showed in [4] (providing full proof in [5]) that the linear upper bound on $\mathbf{F}_{\mathbf{M}}(n)$ can be reduced exponentially — to $\log n + \log\log n + o(\log\log n)$, if the learner is allowed to use programs of general type (from a universal acceptable numbering of all programs) rather than indices in the numbering of the target class. They also established a nearly matching lower bound for the function $\mathbf{F}_{\mathbf{M}}(n)$ (having shown that there exists an indexed class of functions where no strategy can use less than nearly $\log n$ mind changes). In the paper [3], J. Bārzdiņš showed that the lower bound on the number of mind changes jumps to nearly n, if the numbering defining the target class of functions is used as the hypotheses space.

In the paper [6], the authors studied the following problem: is it possible to "speed up" learning of indexed classes of functions achieving as slow growth of the function $\mathbf{F}_{\mathbf{M}}(n)$ as possible? More specifically, if and when is it possible, given any total recursive function $r(n)$ and any learner $\mathbf{M}$ for an indexed class $\mathcal{L}$, to find another learner $\mathbf{M}'$ such that, for all n, $r(\mathbf{F}_{\mathbf{M}'}(n)) \leq \max(\{\mathbf{F}_{\mathbf{M}}(n), c\})$, for some constant c? They suggested to call such a provable statement for a class $\mathcal{L}$ "absolute speed-up theorem" (AST, for brevity), and established validity of AST for any class $\mathcal{L}$ of recursive functions with decidable equivalence problem and not learnable with a constant number of mind changes.

In this paper, we study possibilities of mind change speed-ups in two different contexts. First, we consider **TxtEx**-learning (from all positive data) of indexed families of recursive languages. Secondly, we consider learning in the limit, from positive data, *automatic* classes of languages by *automatic* learners; such an indexed class of languages is defined by a finite automaton (the study of inductive inference in this context was initiated in [15]).

In the general case of **TxtEx**-learning of indexed families, we establish the conditions under which AST is possible: we show that AST holds if (a) the equivalence problem for the languages in the class is decidable and (b) inclusion of one language in another one implies their equality (Theorem 3). Note that the condition (a) typically holds for practically important indexed families of languages (for example, the class of regular languages indexed by finite automata and the class of pattern languages indexed by patterns). In light of this, the condition (b) is really the important criterion deciding if the AST can work for an indexed class. This condition is quite simple and can be typically tested for many practically useful indexed classes. Now, we show that, if the condition (a) holds and the condition (b) does not (and yet there are no subset chains of languages of length more than 2), then $\mathbf{F}_{\mathbf{M}}(n)$ can grow faster than any fixed recursive function (Theorem 6). We also consider the case when the condition (b) holds, but (a) does not. It turns out, that, in this case, any class can be learned with $O(\log n)$ upper bound on $\mathbf{F}_{\mathbf{M}}(n)$ (Theorem 8), and there exists a class with the lower bound of $\log n - o(\log n)$ on $\mathbf{F}_{\mathbf{M}}(n)$ for any learner $\mathbf{M}$ (Theorem 7).

Interestingly, if a learner witnessing AST is required to conjecture grammars only for the languages in the class, then it cannot be made consistent with the input seen so far: for such consistent learners, we show that, for some classes, the lower bound on $\mathbf{F}_{\mathbf{M}}(n)$ is $\log n + 1$ (cf. Theorem 5).

For the automatic case, the definition of $\mathbf{F}_{\mathbf{M}}$ needs to be readjusted, as indices of languages are strings, and the set of indices must be regular; in addition, we require learners to be computable by finite automata (automatic). Accordingly, we consider a (natural) or-

dering of all indices and define $\mathbf{F}_{\mathbf{M}}(w)$ as the maximum of the number of mind changes on all languages with indices not length-lexicographically greater than w with respect to the given ordering.

We have not been able to find a reasonable range of automatic classes for which AST holds. Yet, we obtained some interesting partial results. First, we show that, for any nondecreasing unbounded automatic function, there is an automatic class that can be learned by an automatic learner with $\mathbf{F}_{\mathbf{M}}(w)$ not exceeding this function; yet AST is not possible for this class, as the function provides also the matching lower bound on $\mathbf{F}_{\mathbf{M}}(w)$ (Theorem 10). Then we show that, for a range of automatic classes satisfying a simple condition, $\mathbf{F}_{\mathbf{M}}(w)$ can be made smaller than any unbounded non-decreasing recursive function if an automatic learner uses *fat* texts, where every input datum appears infinitely many times (Theorem 12). This result works also for automatic learners using arbitrary input texts if the languages in the class satisfy the additional condition of being pairwise infinitely different. Although, our results in this section do not directly deal with a more practically interesting AST problem for learning automatic classes with no strong restrictions on either stream of input data or the class to be learnt, they certainly shed light on the difficulties of solving the AST problem without these restrictions.

Mind changes have played an important role in other fields besides inductive inference, such as in computational complexity to determine the powers of Boolean Hierarchy, query order, etc., see for example [7, 14].

2 Preliminaries

Let N denote the set of natural numbers. A language is a subset of N. The symbol $\emptyset$ denotes the empty set. Symbols $\subseteq, \supseteq, \subset, \supset$, respectively, denote subset, superset, proper subset and proper superset. Furthermore, $\max(S), \min(S)$ and $\mathrm{card}(S)$, respectively, denote the maximum, minimum and cardinality of a set S, where $\max(\emptyset) = 0$ and $\min(\emptyset) = \infty$. We use $\mathrm{card}(S) \leq *$ to denote that the cardinality of S is finite.

We let $\langle \cdot, \cdot \rangle$ stand for an arbitrary, computable, one-to-one encoding of all pairs of natural numbers onto N [21]. Similarly, one can define $\langle \cdot, \cdot, \ldots, \cdot \rangle$ coding multiple arguments. We assume these pairing functions to be monotonically increasing in all their arguments.

We let φ denote a fixed *acceptable* programming system for the partial computable functions [21]. The i-th partial computable function in the system φ is denoted by φ_i. The set of all recursive functions is denoted by $\mathcal{R}$. When considering partial computable functions with multiple arguments, we assume that the inputs are coded using the pairing function described above. We let $W_i = \mathrm{domain}(\varphi_i)$.

2.1 Learning Languages in the Limit

A finite sequence σ is a mapping from an initial segment of N into $(N \cup \{\#\})$. We let Λ denote the empty sequence. The content of σ, denoted $\mathrm{content}(\sigma)$, is the set of natural numbers in the range of σ. The length of σ, denoted $|\sigma|$, is the number of elements in the domain of σ. SEQ denotes the set of all finite sequences. A text T is a mapping from N to $(N \cup \{\#\})$. The content of T, denoted $\mathrm{content}(T)$, is the set of natural numbers in the range of T. A text T is for a language L iff $\mathrm{content}(T) = L$. $T[n]$ denotes the initial segment of T of length n, and $\sigma[n]$ denotes the initial segment of σ of length n. Intuitively, #'s denote pauses in the presentation of data. A text T is called *fat* [20] if for every $x \in \mathrm{content}(T)$, there exist infinitely many n such that $T(n) = x$.

A language learning machine is an algorithmic mapping from SEQ to $N \cup \{?\}$. Intuitively, ? denotes that the learner does not have enough data to form a conjecture. We let $\mathbf{M}$, with or without decorations, range over learning machines. If, for all but finitely many n, $\mathbf{M}(T[n]) = i$, then we say that $\mathbf{M}(T)\downarrow = i$ (or simply, $\mathbf{M}(T) = i$). If there exists an i such that $\mathbf{M}(T)\downarrow = i$, then we say that $\mathbf{M}(T)$ converges (written: $\mathbf{M}(T)\downarrow$); otherwise, we say that $\mathbf{M}(T)$ diverges or $\mathbf{M}(T)$ is undefined (written: $\mathbf{M}(T)\uparrow$).

▶ Definition 1. [13] (a) $\mathbf{M}$ **TxtEx**-identifies a language L (written: $L \in \mathbf{TxtEx}(\mathbf{M})$) iff for all texts T for L, $\mathbf{M}(T)\downarrow$ and $W_{\mathbf{M}(T)} = L$.
(b) $\mathbf{M}$ **TxtEx**-identifies a class $\mathcal{L}$ of languages iff $\mathbf{M}$ **TxtEx**-identifies each $L \in \mathcal{L}$.
(c) $\mathbf{TxtEx} = \{\mathcal{L} : (\exists \mathbf{M})[\mathbf{M} \text{ } \mathbf{TxtEx}\text{-identifies } \mathcal{L}]\}$.

For a learner $\mathbf{M}$, a text T, and $n \in N$, we let $\mathrm{MC}_{\mathbf{M}}(T[n])$ denote the number of mind changes [9, 8] made by $\mathbf{M}$ on $T[n]$, that is, $\mathrm{card}(\{r < n : ? \neq \mathbf{M}(T[r]) \neq \mathbf{M}(T[r+1])\})$. Similarly, $\mathrm{MC}_{\mathbf{M}}(T)$ denotes the number of mind changes [9, 8] made by $\mathbf{M}$ on T, that is, $\mathrm{card}(\{r : ? \neq \mathbf{M}(T[r]) \neq \mathbf{M}(T[r+1])\})$. We let $\mathrm{MC}_{\mathbf{M}}(L)$ denote the maximum over $\mathrm{MC}_{\mathbf{M}}(T)$ for all texts T for L. One can assume without loss of generality that, if $\mathbf{M}(\sigma) \neq ?$ and $\sigma \subseteq \tau$, then $\mathbf{M}(\tau) \neq ?$.

A learner $\mathbf{M}$ is said to be *consistent* [1, 2] if for all $\sigma \in \mathrm{SEQ}$, $\mathrm{content}(\sigma) \subseteq W_{\mathbf{M}(\sigma)}$.

An indexed family is a family $\mathcal{L} = (L_i)_{i \in N}$ of languages such that, $\{(i, x) : x \in L_i\}$ is recursive. When dealing with indexed families, we let $\mathbf{F}_{\mathbf{M}}(i) =$ maximum over $\mathrm{MC}_{\mathbf{M}}(T)$ on any input text T for a language L_j, $j \leq i$.

Often, when learning indexed families, instead of using the acceptable programming system $W_0, W_1, \ldots$ as hypothesis space, we use an indexed family, $(H_i)_{i \in N}$, as hypothesis space. That is, in Definition 1(a), we require $H_{\mathbf{M}(T)} = L$, instead of requiring $W_{\mathbf{M}(T)} = L$. This model of learning is said to be *class preserving* [18, 23] if $\{H_i : i \in N\} = \{L_i : i \in N\}$. In theorems in the sequel, for positive learnability statements, by default, we take the hypothesis space $H_i = L_i$, unless specified otherwise. For non-learnability statements, we allow acceptable programming system $(W_i)_{i \in N}$ as hypothesis space, (and thus the diagonalization works against arbitrary hypothesis spaces).

We now formally define AST.

▶ Definition 2. Suppose an indexed family $\mathcal{L} = (L_i)_{i \in N}$ is given. We say that $\mathcal{L}$ satisfies *absolute speed-up theorem* (AST) if for any recursive function $r(\cdot)$ and a learner $\mathbf{M}$ for $\mathcal{L}$, there exists another learner $\mathbf{M}'$ and a constant c such that, for all n, $r(\mathbf{F}_{\mathbf{M}'}(n)) \leq \max(\{\mathbf{F}_{\mathbf{M}}(n), c\})$.

3 Mind Change Speed-up for Learning Recursive Languages

Our main goal in this section is to establish conditions under which AST holds for learning an indexed class of languages. First note that AST does not hold for some indexed classes. This follows from Theorem 9 below, for automatic families. The following theorem gives conditions for AST holding for an indexed class (the actual AST is stated in the corollary).

Proof of the following theorem essentially uses the idea of delaying mind change until it is safe, that is, until all grammars, except for at most one grammar, upto a sufficiently large bound are found to be incompatible with the input data.

▶ Theorem 3. Suppose $\mathcal{L} = (L_i)_{i \in N}$ is an indexed family for which the equivalence problem is decidable. Furthermore, assume that $L_i \subseteq L_j$ implies $L_i = L_j$. Suppose h is a monotonically non-decreasing recursive function, with range(h) being unbounded.

Then, there exists a learner $\mathbf{M}$ which **TxtEx**-learns $\mathcal{L}$ such that $\mathbf{F}_{\mathbf{M}}(n) \leq h(n)$.

Proof. Let $H(k) = \min(\{k' : h(k') > k\})$. Let $\mathbf{M}(\Lambda) =?$. Inductively, define $\mathbf{M}(T[n+1])$ as follows.

If for all $j \leq n$, $\text{content}(T[n+1]) \not\subseteq L_j$, then let $\mathbf{M}(T[n+1]) = \mathbf{M}(T[n])$.

Otherwise, let j be least such that $\text{content}(T[n+1]) \subseteq L_j$. If there exists a $j' < H(\text{MC}_{\mathbf{M}}(T[n]) + 1)$, such that $L_j \neq L_{j'}$ (this can be tested, as the equivalence problem is decidable) and $\text{content}(T[n+1]) \subseteq L_{j'}$, then let $\mathbf{M}(T[n+1]) = \mathbf{M}(T[n])$ (the learner $\mathbf{M}$ "does not want" to change mind to j, as there is a different language containing the same initial segment of input data not "too far" from j — as defined by the function H); otherwise let $\mathbf{M}(T[n+1]) = j$.

Note that if $\mathbf{M}(T[n+1]) = j$, then for all $j' < H(\text{MC}_{\mathbf{M}}(T[n]) + 1)$, $L_j = L_{j'}$ or $\text{content}(T[n+1]) \not\subseteq L_{j'}$. That is, for all j' such that $h(j') \leq \text{MC}_{\mathbf{M}}(T[n]) + 1$, $L_j = L_{j'}$ or $\text{content}(T[n+1]) \not\subseteq L_{j'}$. Thus, if $\text{content}(T[n+1]) \subseteq L_i$ for an L_i different from L_j, i must be so large that $\text{MC}_{\mathbf{M}}(T[n+1]) < h(i)$. It follows that, given any L_i, for any text T for L_i, $\text{MC}_{\mathbf{M}}(T) \leq h(i)$. Furthermore, $\mathbf{M}$ **TxtEx**-identifies L_i on a text T for L_i, as after it has received $T[n+1]$ such that $\text{content}(T[n+1]) \not\subseteq L_{j'}$ for any j' such that $h(j') \leq \max(\{h(i), 1\})$, we will have $\mathbf{M}(T[n+1]) = i$. ◀

▶ Corollary 4. Suppose $\mathcal{L} = (L_i)_{i \in N}$ is an indexed family for which the equivalence problem is decidable. Furthermore, assume that $L_i \subseteq L_j$ implies $L_i = L_j$. Then, AST holds for $\mathcal{L}$.

Proof. Suppose $\mathbf{M}$ **TxtEx**-identifies $\mathcal{L}$ and $\mathbf{F}_{\mathbf{M}}$ is the corresponding mind change complexity. The corollary is trivial if $\mathbf{F}_{\mathbf{M}}$ is bounded by a constant. So assume $\mathbf{F}_{\mathbf{M}}$ is unbounded. Given a recursive function r, define the recursive function h such that, $h(0) = 0$, and $h(n+1) = h(n)+1$ if $r(h(n)+1) \leq \mathbf{F}_{\mathbf{M}}(n+1)$ as can be verified by running $\mathbf{M}$ on some σ of length at most n, such that $\text{content}(\sigma) \subseteq \{x : x \leq n\} \cap L_i$, for some $i \leq n$; $h(n+1) = h(n)$ otherwise. Thus, $r(h(n)) \leq \mathbf{F}_{\mathbf{M}}(n)$. Now the corollary follows from Theorem 3. ◀

The above corollary immediately gives the result of Bārzdiņš, Kinber and Podnieks [6] that, in the case of inductive inference of indexed classes of recursive functions, decidability of the equivalence problem for the functions in an indexed class suffices for AST.

It can be easily shown that the conditions of Theorem 3 are not necessary — for example, one can easily transform any indexed class $\mathcal{L} = (L_i)_{i \in N}$ satisfying the conditions of Theorem 3 into a class $\mathcal{L}' = (L_j)'_{j \in N}$ with undecidable equivalence problem and AST holding for it. For this, one takes either $L'_{2j} = L'_{2j+1} = \{2x : x \in L_j\}$ or $L'_{2j} = \{2x : x \in L_j\} \cup \{2 * \langle 2j, r_j \rangle + 1\}$ and $L'_{2j+1} = \{2x : x \in L_j\} \cup \{2 * \langle 2j+1, r_j \rangle + 1\}$, for some appropriate large enough r_j, such that the i-th Turing Machine does not correctly decide whether $L'_{2j} = L'_{2j+1}$.

Note that the learner in the proof of Theorem 3 can be made consistent (for indexed families), if the learner is allowed to output N as a conjecture. For this, if the conjecture of the above learner is inconsistent (including for the initial conjecture ?), then it is replaced by a conjecture for N. This doubles the number of mind changes made, however this problem can be easily addressed by replacing $h(i)$ by $\lfloor (h(i) \dot{-} 1)/2 \rfloor$ in the above construction. Then, the above result holds even for consistent learners, for any non-decreasing unbounded recursive h which is ≥ 1 on all inputs. However, the consistent learner outputting N from time to time may not be class preserving. In case one requires class preserving consistency, the following theorem holds.

▶ Theorem 5. Suppose $L_i = \{\langle x, b_x \rangle : x \in N\}$, where b_r is the $(r+1)$-th least significant bit of i in binary representation (the least significant bit is b_0).
Let $\mathcal{L} = \{L_i : i \in N\}$. Then,

(a) $\mathcal{L}$ can be class-preservingly consistently learnt by a learner $\mathbf{M}$ which makes at most $\lceil \log(i+1) \rceil$ mind changes on L_i;
(b) For any class-preserving consistent learner $\mathbf{M}$ for $\mathcal{L}$, $\mathbf{F}_{\mathbf{M}}(n) \geq \lceil \log(n+1) \rceil$.

Now we consider what happens if the conditions of Theorem 3 do not hold. First, we consider the case when decidability of the equivalence problem still holds, but subset chains of length more than 1 are allowed.

Proof of the following Theorem 6 essentially exploits the following idea. Note that for any infinite set B and finite sequence σ, if content$(\sigma) \subseteq B$, and a learner learns both B and B', a finite subset of B containing content(σ), then the learner makes a mind change, beyond σ, on some text for B extending σ. For each learner $\mathbf{M}_i$ the proof uses a set L_{2i} (representing B above). It then constructs $\sigma_0, \sigma_1, \ldots, \sigma_r$, with $r \leq h(2i)$, by potentially placing a finite subset of L_{2i} containing content(σ_j) into the class $\mathcal{L}$ in order to force $h(2i)$ mind changes by $\mathbf{M}_i$ (in case $\mathbf{M}_i$ learns $\mathcal{L}$). It will be the case that at most one of the above finite sets is actually placed in $\mathcal{L}$ and others are spoiled (by making them non-subset of L_{2i}), thus satisfying the requirement of having a subset chain of length at most 2.

▶ Theorem 6. Suppose h is any recursive increasing function. There exists an indexed family $\mathcal{L}$, where the indexing is one-to-one, for which there is no subset chain of length more than 2, and there is no speedup. That is,
(a) $\mathcal{L}$ can be **TxtEx**-learnt, using a class preserving hypothesis space, by a learner $\mathbf{M}$, such that $\mathbf{F}_{\mathbf{M}}(i) \leq h(i)$.
(b) For any $\mathbf{M}$ which **TxtEx**-identifies $\mathcal{L}$, $\mathbf{F}_{\mathbf{M}}(2i) \geq h(2i)$.

Proof. For ease of notation we assume that $h(0) \geq 1$. Let $L_{2i} = \{\langle i, x \rangle : x \text{ is odd}\}$.

Let $A_i^r = \{\langle i, x \rangle : x \text{ is odd and } x \leq r\}$.

Let $B_i^{r,y} = \{\langle i, x \rangle : x \text{ is odd and } x \leq r\} \cup \{\langle i, 2\langle r, y \rangle \rangle\}$.

We let $\mathcal{L} = \{L_j : j \in N\}$, where L_j, for odd j, are defined below. They will be of the form A_i^r or $B_i^{r,y}$ which are chosen to be in the class based on the following construction (where it can be easily ensured that if $L_j = A_i^r$ or $B_i^{r,y}$, then $j \geq 2i$).

It will be the case that, for any i, r, $\mathcal{L}$ contains at most one of A_i^r or $B_i^{r,y}$ (for some y), and there are at most $h(2i)$ many different r's such that $\mathcal{L}$ contains A_i^r or $B_i^{r,y}$ (for some y).

We now give the construction for L_j, for j being odd. The following process is run in dovetailing fashion for each i. For a given i, the languages constructed below are of the form A_i^r or $B_i^{r,y}$. These are used to diagonalize against the learner $\mathbf{M}_i$.

Whenever the process below needs to define a new language (L_j), we assume that a $j \geq 2i$ is allocated in some fashion so that all L_j, j being odd, get defined when one considers the processes for different i; note that for every i at least one language gets defined below.

Construction for the languages in $\mathcal{L}$ which are of the form A_i^r or $B_i^{r,y}$.

Initially, let $\sigma_0 = \langle i, 1 \rangle$.

For $k = 0$ to $h(2i) - 1$ do:

1. Let $w = \max(\{w' : \langle i, w' \rangle \in \text{content}(\sigma_k)\})$.
2. Add a new language, say L_j, to $\mathcal{L}$. Initially, L_j is A_i^w. Define more and more elements not in A_i^w to be not in L_j until step 3 succeeds. If and when step 3 succeeds, go to step 4.
3. Search for a τ extending σ_k such that content$(\tau) \subseteq L_{2i}$, and $\mathbf{M}_i(\sigma_k) \neq \mathbf{M}_i(\tau)$.
4. Let $L_j = B_i^{w,y}$, for an even y such that $L_j(\langle i, 2\langle w, y \rangle \rangle)$ has not been defined upto now and $y > w$.
5. Let σ_{k+1} be an extension of τ such that content$(\sigma_{k+1}) = A_i^{w'}$, for some odd $w' > w$ such that w' bounds the time needed to get upto here in the construction.

EndFor

Note that if $\mathbf{M}_i$ **TxtEx**-learns $\mathcal{L}$, then the search in step 3 will succeed. Furthermore, only the last incomplete iteration of the "for" loop may generate a subset of L_{2i}. All other languages generated are incomparable to each other. Thus the languages in $\mathcal{L}$ satisfy the "subset" constraints of the theorem.

Furthermore, if $\mathbf{M}_i$ **TxtEx**-learns $\mathcal{L}$, then the above construction forces at least $h(2i)$ mind changes for $\mathbf{M}_i$ on some text for L_{2i}. Thus, the condition (b) of the theorem holds.

To see (a), let $g(i,k)$ denote a program which decides L_j, for the j as in iteration k of the for loop above if iteration k exists; otherwise it is a program which decides L_{2i}. Note that such a grammar can be easily defined, as one can slowly follow L_{2i}, and if one observes iteration k to have started, then follow L_j as in there — see step 5 in the construction above which allows us to do this.

Now, if $\langle i, w\rangle$ is the largest element seen in the input so far and w is even, then the learner immediately knows the input language and can output a grammar appropriately. On the other hand, if w is odd, then the learner simulates the construction (for parameter i) above for w steps to find the largest k such that the construction above, after w steps, reaches iteration k in the loop. Now, if $\langle i, w\rangle$ belongs to L_j, where j is as in iteration k of the loop in the construction above, then the learner outputs $g(i,k)$. Otherwise, it outputs $g(i,k+1)$.

It is easy to verify that the learner above **TxtEx**-learns the class $\mathcal{L}$ and makes at most $h(k)$ mind changes on a text for the language L_k. ◀

Now we will study what can happen if the languages in an indexed class are equal or incomparable, but the equivalence problem may be undecidable. Proofs of the next two theorems are based on techniques used in [5, 11] for similar theorems for function learning.

▶ Theorem 7. Given any non-decreasing recursive function f with unbounded range, there exists an indexed family $\mathcal{L} = (L_i)_{i\in N}$, where, for all j and k, either $L_j = L_k$ or L_j and L_k are incomparable, such that for all $\mathbf{M}$ **TxtEx**-identifying $\mathcal{L}$, $\mathbf{F}_{\mathbf{M}}(n) \geq \log n - f(n)$ for infinitely many n.

The next theorem shows that, yet, every indexed class with equal or incomparable languages can be learned using approximately $\log n$ mind changes.

▶ Theorem 8. Every indexed family $\mathcal{L} = (L_i)_{i\in N}$, such that for all i, j, either $L_i = L_j$ or L_i and L_j are incomparable, can be **TxtEx**-learnt by a learner $\mathbf{M}$, using a class preserving hypothesis space, such that $\mathbf{F}_{\mathbf{M}}(n) \leq \log n + \log\log n + o(\log\log n)$.

(Here, for ease of notation, we take $\log n$ and $\log\log n$ to be 1, for $n \leq 2$).

One can improve the bound in the above theorem to $\mathbf{F}_{\mathbf{M}}(n) \leq \log n + \log\log n + \ldots + o(\log\log\log\ldots\log n)$. Note that the above result does not hold if one requires that the learner uses the given indexing of $\mathcal{L}$ as the hypothesis space. This follows from the corresponding result for function learning from [3].

4 Automatic Classes and Learning

In this section, we introduce necessary concepts for automatic learning of automatic classes. Let Σ denote a non-empty finite alphabet. Let Σ^* denote the set of all strings over the alphabet Σ. Let ϵ denote the empty string. We let $|w|$ denote the length of string w. We fix some arbitrary order among the members of Σ. For strings x and y, $x <_{lex} y$ denotes that x is lexicographically (that is, in dictionary order) before y. The relation $x <_{ll} y$ denotes that x is length-lexicographically before y, that is, either $|x| < |y|$, or $|x| = |y|$ and $x <_{lex} y$. When we consider sets of strings, $\min(S)$ and $\max(S)$ denote the length-lexicographically minimal

and maximal strings in S, where $\max(\emptyset) = \epsilon$ and $\min(\emptyset)$ is undefined. We let $\text{succ}_L(w)$ and $\text{pred}_L(w)$ denote the successor and predecessor of w in the length-lexicographical ordering of the language L, where $\text{pred}_L(w)$ is undefined for the length-lexicographically least string in L, and $\text{succ}_L(w)$ is undefined for the length-lexicographically maximal string in L (if any). For a given Σ and $w \in \Sigma^*$, let $ord(w)$ denote the number of strings in Σ^* which are $<_{ll} w$. We let cf_L denote the characteristic function of L.

The *convolution* (see [17]) of two strings $x, y \in \Sigma^*$, $conv(x, y)$, is defined as the string $(x(0), y(0))(x(1), y(1)) \ldots (x(n-1), y(n-1))$, where each pair is a symbol from $(\Sigma \cup \{\diamond\})^2$ and $n = \max(|x|, |y|)$. The special symbol $\diamond \notin \Sigma$ is appended (as many times as needed) to the shorter string in order to make both strings to be of the same length n. Similarly, *conv* can be defined on multiple arguments. An n-ary relation R or an m-ary function f is called *automatic* if the sets $\{\text{conv}(x_1, x_2, \ldots, x_n) : R(x_1, x_2, \ldots, x_n)\}$ and $\{\text{conv}(x_1, x_2, \ldots, x_m, y) : f(x_1, x_2, \ldots, x_m) = y\}$, respectively, are regular.

A family of languages over alphabet Σ, $\{L_\alpha : \alpha \in I\}$ is said to be *automatic* (see [17]) iff I is a regular set, each $L_\alpha \subseteq \Sigma^*$, and $\{\text{conv}(\alpha, x) : x \in L_\alpha\}$ is regular. When we are considering learning of automatic classes, the elements of languages are strings rather than natural numbers. Most of the definitions and notations discussed above for learning languages over natural numbers carry over to the case of learning languages over strings, with numbers being replaced by strings; we omit the details. Below we describe a special kind of learner, called *automatic* learner ([15]). An automatic learner is an automatic mapping from previous memory, current datum to new memory and new conjecture. Here memory is a string over some alphabet Γ. Suppose T is the input text for the automatic learner $\mathbf{Q}$. Let $(mem^T_{n+1}, hyp^T_{n+1}) = \mathbf{Q}(mem^T_n, T(n))$, where mem^T_0 and hyp^T_0 are some default initial memory mem_0 and the default initial hypothesis hyp_0 of the learner $\mathbf{Q}$. We can consider the hypothesis hyp^T_n of the learner $\mathbf{Q}$ as its output on the input $T[n]$, and thus the learnability notions discussed in Section 2.1 above can be taken over to the setting of automatic learners. Below we let $\mathbf{Q}$ range over automatic learners.

When dealing with automatic families, we let $\mathbf{F}_{\mathbf{M}}(w) =$ maximum over the mind changes made by the learner $\mathbf{M}$ on any input text for a language L_u, $u \leq_{ll} w$. Note that for learning automatic families, as long as memory is not restricted (except due to the definition of automatic learner), one can assume the hypothesis space to be the same as the automatic class being learnt. Thus, for the next section, for all the results the hypothesis space used is the automatic family being learnt.

5 Mind Change Speed-up for Automatic Classes

In the sequel, pairing is assumed to be done via convolution. We begin with an example of an automatic class containing languages over the unary alphabet with linear lower and upper bounds on the number of mind changes.

▶ Theorem 9. Let $L_{0^i} = \{0^j : j < i\}$. Let $\mathcal{L} = \{L_{0^i} : i \in N\}$. Then,
(a) $\mathcal{L}$ can be **TxtEx**-learnt by an automatic learner $\mathbf{Q}$ such that $\mathbf{F}_{\mathbf{Q}}(0^n) = n$.
(b) Any learner $\mathbf{M}$ which **TxtEx**-learns $\mathcal{L}$ has $\mathbf{F}_{\mathbf{M}}(0^n) \geq n$.

Proof. Consider an iterative learner $\mathbf{Q}$ which starts with conjecture 0^0. $\mathbf{Q}$, on previous conjecture 0^j and new input 0^i, outputs $0^{\max(\{i+1,j\})}$. Clearly, $\mathbf{Q}$ satisfies (a). Part (b) follows easily as for any **TxtEx**-learner for $\mathcal{L}$, one can construct σ_i, $i \in N$, such that $\sigma_i \subseteq \sigma_{i+1}$, $\text{content}(\sigma_i) = L_{0^i}$, and $\mathbf{M}(\sigma_i)$ is a grammar for L_{0^i}. Then $\text{MC}_{\mathbf{M}}(\sigma_i) \geq i$. ◀

Now we show that, for any automatic function h (with the range containing strings over a unary alphabet), there is an automatic class that can be learned automatically with h (more precisely, $ord(h(0^{i+1},\epsilon)+1)$ being the tight bound on the number of mind changes.

▶ Theorem 10. Suppose h is a non-decreasing automatic function with range$(h) \subseteq 0^+$. Let $L_{(0^{i+1},\epsilon)} = \{(0^{i+1},1^j) : j \in N\}$, $L_{(0^{i+1},1^{j+1})} = \{(0^{i+1},1^r) : r < j+1\}$, $L_{(\epsilon,\epsilon)} = \emptyset$, and $\mathcal{L} = \{L_{(\epsilon,\epsilon)}\} \cup \{L_{(0^{i+1},1^j)} : i \in N, j \leq ord(h(0^{i+1},\epsilon))\}$. Then,
(a) $\mathcal{L}$ can be **TxtEx**-learnt by an automatic learner $\mathbf{Q}$, such that $\mathbf{F_Q}(0^{i+1},\epsilon) = ord(h(0^{i+1},\epsilon)) + 1$.
(b) Any learner $\mathbf{M}$ which **TxtEx**-learns $\mathcal{L}$ has $\mathbf{F_M}(0^{i+1},\epsilon) \geq ord(h(0^{i+1},\epsilon)) + 1$.

Now our goal is to show that, under certain natural conditions, mind change speed-up for automatic classes is possible if an automatic learner uses fat texts. Proofs of Theorems 11 and 12 are the most difficult in this paper. In these theorems, on one hand, the class $\mathcal{L}$ considered is automatic (so equivalence, subset problem, etc., among languages in the class are decidable), but, on the other hand, the learner is automatic and we also allow some subset relations among languages. The main difficulty is because of the learner being automatic, thus forgetting past data. The proof again uses delaying of mind change until it is safe, by cancelling all but c wrong grammars upto some large enough bound (in a way similar to Proof for Theorem 3). Here c is a constant such that at most c different languages in the class are related by subset/superset relation with any particular language of the class. Then, the learner finds upto c^2 many grammars which may be for the input language, in case any of the languages, with indices below the large enough bound mentioned above, contains the input language. The learner then proceeds to try these languages one by one (where smaller languages are tried first). Due to forgetting of past data by automatic learners, one needs a fat text to be able to cancel out wrong grammars. The proof for arbitrary speed-up (Theorem 12) is technically involved, and thus we begin by showing a simpler version first.

▶ Theorem 11. Suppose $\mathcal{L} = \{L_\alpha : \alpha \in I\}$ is an automatic family (without loss of generality, assume one-to-one). Suppose constants k and c are given, where for all $L \in \mathcal{L}$, card$(\{L' \in \mathcal{L} : L \subseteq L' \text{ or } L' \subseteq L\}) \leq c$.

Then, there exists an automatic learner $\mathbf{Q}$ which learns $\mathcal{L}$ from fat texts such that (for learning from fat texts) $\mathbf{F_Q}(\alpha) \leq \max(\{\lceil |\alpha|/k \rceil, 1\}) * c^2 - 1$.

Proof. Without loss of generality assume that there are at least $c+1$ indices of length at most k. The learner Q defined below operates in phases. Intuitively, memory of $\mathbf{Q}$ is of the form $(0^i, 0^p, \alpha_1, \alpha_2, \ldots, \alpha_{c+1}, \beta_1, \beta_2, \ldots, \beta_{c^2}, prevconj)$, where

- (i) $p = k * i$;
- (ii) $\alpha_j <_{ll} \alpha_{j+1}$, for $1 \leq j < c$;
- (iii) $\alpha_c \leq_{ll} \alpha_{c+1}$;
- (iv) $prevconj$ is the previous conjecture;
- (v) $\mathbf{Q}$ has already made $(i-1)$-phases (each producing upto c^2 conjectures), and is now in its i-th phase;
- (vi) for all α such that $|\alpha| \leq p$ and $\alpha \notin \{\alpha_j : 1 \leq j \leq c\} \cup \{\gamma : \alpha_c \leq_{ll} \gamma \leq_{ll} \alpha_{c+1}\}$, $\mathbf{Q}$ has already observed a string in the input which is not in L_α;
- (vii) in case $\alpha_c = \alpha_{c+1}$, $\beta_1, \ldots, \beta_{c^2}$ denote the c^2 possible members β of I such that L_β is contained in one of L_{α_j}, $1 \leq j \leq c$ (in case of $< c^2$ such members, we use # for the remaining elements); furthermore, if $L_{\beta_j} \subseteq L_{\beta_{j'}}$, then $j \leq j'$;
- (viii) in case $\alpha_c = \alpha_{c+1}$, $prevconj = \beta_j$ for some j such that $1 \leq j \leq c^2$, and for $1 \leq j' < j$, $\mathbf{Q}$ has already observed a string in the input which is not in $L_{\beta_{j'}}$.

Initially, the memory of $\mathbf{Q}$ is $(0^1, 0^k, \alpha_1, \alpha_2, \alpha_3, \ldots, \alpha_c, \alpha_{c+1}, \#, \#, \ldots, \#, ?)$, where α_{c+1} is the length-lexicographically largest element of I of length at most k, and $\alpha_1, \ldots, \alpha_c$ are the c length-lexicographically least elements of I. The initial conjecture of $\mathbf{Q}$ is ?.

At any point during the learning process, if the new input is w and the previous conjecture is $(0^i, 0^p, \alpha_1, \alpha_2, \ldots, \alpha_{c+1}, \beta_1, \beta_2, \ldots, \beta_{c^2}, prevconj)$, then $\mathbf{Q}$ behaves as follows:

- (1.) If $\alpha_c \neq \alpha_{c+1}$, and $w \notin L_{\alpha_j}$ for some least j with $1 \leq j \leq c+1$, then
  - (1.1.) If $j = c+1$, then let $\alpha'_{c+1} = \text{pred}_I(\alpha_{c+1})$, and $\alpha'_r = \alpha_r$ for $1 \leq r \leq c$; otherwise, let $\alpha'_r = \alpha_r$, for $1 \leq r < j$, $\alpha'_r = \alpha_{r+1}$, for $j \leq r < c$, and $\alpha'_c = \text{succ}_I(\alpha_c)$.
  - (1.2.) If $\alpha'_c \neq \alpha'_{c+1}$, then let $\beta_1 = \ldots = \beta_{c^2} = \#$, and let new memory be $(0^i, 0^p, \alpha'_1, \ldots, \alpha'_{c+1}, \beta_1, \ldots, \beta_{c^2}, prevconj)$ and let new conjecture be $prevconj$.
  - (1.3.) else (i.e., $\alpha'_c = \alpha'_{c+1}$), let $\beta_1, \ldots, \beta_{c^2}$ denote the c^2 possible members β of I such that L_β is contained in one of $L_{\alpha'_j}$, $1 \leq j \leq c$; furthermore, if $L_{\beta_j} \subseteq L_{\beta_{j'}}$, then $j \leq j'$; If there are several possible orders to choose β_j satisfying the above, then choose the lexicographically least order among them. (In case of $< c^2$ members β of I such that L_β is comparable to some L_{α_j}, we use # for the remaining β's); Conjecture β_1, and let new memory be $(0^i, 0^p, \alpha'_1, \ldots, \alpha'_{c+1}, \beta_1, \ldots, \beta_{c^2}, \beta_1)$.
- (2.) else (if $\alpha_c = \alpha_{c+1}$), then
  - if $w \notin L_{prevconj}$, then
    - (2.1.) if $prevconj = \beta_j$, and $j < c^2$ and $\beta_{j+1} \neq \#$, then let new memory be $(0^i, 0^p, \alpha_1, \alpha_2, \ldots, \alpha_{c+1}, \beta_1, \beta_2, \ldots, \beta_{c^2}, \beta_{j+1})$ and the new conjecture be β_{j+1}.
    - (2.2.) otherwise, let new memory be
      $$(0^{i+1}, 0^{p+k}, \alpha'_1, \alpha'_2, \ldots, \alpha'_{c+1}, \#, \#, \ldots, \#, prevconj),$$
      where α'_{c+1} is the length-lexicographically largest element of I of length at most $p+k$, and $\alpha'_1, \ldots, \alpha'_c$ are the c length-lexicographically least elements of I.
  - else (i.e., $w \in L_{prevconj}$) repeat the old memory and conjecture.
- (3.) else (i.e., $\alpha_c \neq \alpha_{c+1}$, and $w \in L_{\alpha_j}$ for all j with $1 \leq j \leq c+1$) repeat the old memory and conjecture.

Intuitively, for any w, in step (1) the learner (over several inputs) tries to eliminate all but c of the potential conjectures of length at most p; all the eliminated conjectures do not contain the input language (see steps 1, 1.1 and 1.2). Once the learner is left with only c conjectures of length at most p, which may contain the input language, it finds the indices of all the potential c^2 many languages which may be for the input language (unless none of the languages, with index of length at most p, contain the input language) (see step 1.3).

After this, in steps 1.3, 2 and 2.1, the learner serially tries all the above c^2 many languages which could be the input language. (Note that, the testing of these languages is done in a specific order so that subsets are tried earlier than the supersets.) Then, the learner eliminates them one by one, until it finds the correct language or observes that none of them could contain the input language (i.e., all languages in $\mathcal{L}$ which contain the input have indices of length larger than p). In which case the learner goes to the next $(i+1$-th) phase (step 2.2).

It is now easy to verify that the above learner **TxtEx**-identifies $\mathcal{L}$ on fat texts, and on L_α makes at most $\max(\{\lceil |\alpha|/k \rceil, 1\}) * c^2 - 1$ mind changes (using $\max(\{\lceil |\alpha|/k \rceil, 1\})$ phases, each of which may make upto c^2 conjectures). ◀

Note that the above proof uses fat texts to be able to check whether a language in the automatic family contains the input language or not. In the above theorem, one can replace c^2 by c, if, instead of using conjectures β_j one by one, the learner (i) keeps track of β_j such

that it hasn't seen a non-element of L_{β_j}, and (ii) outputs a conjecture β_j if the learner hasn't seen a non-element of L_{β_j} and L_{β_j} is contained in every other $L_{\beta_{j'}}$ for which it hasn't seen a non-element. This ensures that in steps 1.3 and 2, for each i, at most c conjectures are output.

Furthermore, we can generalize the theorem above to beat (almost everywhere) mind changes given by any non-decreasing unbounded recursive function as follows.

▶ Theorem 12. Suppose $\mathcal{L} = \{L_\alpha : \alpha \in I\}$ is an automatic family (without loss of generality assume one-to-one). Suppose a non-decreasing unbounded recursive function h and a constant c are given, where for all $L \in \mathcal{L}$, $\text{card}(\{L' \in \mathcal{L} : L \subseteq L' \text{ or } L' \subseteq L\}) \leq c$. Then, there exists an automatic learner $\mathbf{Q}$ which learns $\mathcal{L}$ from fat texts such that (for learning from fat texts) $\mathbf{F}_{\mathbf{Q}}(\alpha) \leq \max(\{h(|\alpha|), 1\}) * c - 1$.

The above result also works if, instead of using fat texts, the languages in the class are required to be pairwise infinitely different or the alphabet size is 1 (in addition to the requirement: for all $L \in \mathcal{L}$, $\text{card}(\{L' \in \mathcal{L} : L \subseteq L' \text{ or } L' \subseteq L\}) \leq c$, for some constant c).

6 Conclusion

In 1972, Bārzdiņš and Freivalds introduced the maximum number of mind changes on the first n functions as a measure of efficiency of learning in the limit. Our interest in this measure of complexity for learning indexed classes of languages was revived by growing interest in automatic learning of automatic classes of languages. As mind change speed-up effects, discussed and resolved for learning recursive functions in [6], surprisingly, have never been explored for learning languages from positive data, we, first, considered these issues for the corresponding framework. We also give a sufficient condition for a family of automatic classes for which speed-up is possible if either an automatic learner uses fat texts, or the languages in the classes in question differ infinitely. Yet the general problem of whether there are wide natural automatic classes for which mind change speed-up is possible remains open.

One can note that the mind change speed-up in both frameworks considered in our paper is achieved when a learner, choosing a new conjecture, accesses increasingly more data from the underlying numbering of languages. It would be very interesting to find out if the amount of such data can be measured in some form and what is the actual quantitative relationship between this amount and the number of mind changes.

Acknowledgements We thank the anonymous referees for several helpful comments.

References

1 D. Angluin. Finding patterns common to a set of strings. *Journal of Computer and System Sciences*, 21(1):46–62, 1980.

2 J. Bārzdiņš. Inductive inference of automata, functions and programs. In *Proceedings of the 20th International Congress of Mathematicians, Vancouver*, pages 455–460, 1974. In Russian. English translation in American Mathematical Society Translations: Series 2, 109:107-112, 1977.

3 J. Bārzdiņš. Limiting synthesis of τ numbers. In *Theory of Algorithms and Programs, vol. 1*, pages 112–116. Latvian State University, Riga, Latvia, 1974. In Russian.

4 J. Bārzdiņš and R. Freivalds. On the prediction of general recursive functions. *Soviet Mathematics Doklady*, 13:1224–1228, 1972.

5 J. Bārzdiņš and R. Freivalds. Prediction and limiting synthesis of recursively enumerable classes of functions. In *Theory of Algorithms and Programs, vol. 1*, pages 101–111. Latvian State University, Riga, Latvia, 1974. In Russian.

6 J. Bārzdiņš, E. Kinber, and K. Podnieks. Concerning synthesis and prediction of functions. In *Theory of Algorithms and Programs, vol. 1*, pages 117–128. Latvian State University, Riga, Latvia, 1974. In Russian.

7 R. Beigel. Bounded queries to SAT and the boolean hierarchy. *Theoretical Computer Science*, 84:199–223, 1991.

8 J. Case and C. Lynes. Machine inductive inference and language identification. In M. Nielsen and E. M. Schmidt, editors, *Proceedings of the 9th International Colloquium on Automata, Languages and Programming*, volume 140 of *Lecture Notes in Computer Science*, pages 107–115. Springer-Verlag, 1982.

9 J. Case and C. Smith. Comparison of identification criteria for machine inductive inference. *Theoretical Computer Science*, 25:193–220, 1983.

10 R. Daley and C. Smith. On the complexity of inductive inference. *Information and Control*, 69:12–40, 1986.

11 R. Freivalds, J. Bārzdiņš, and K. Podnieks. Inductive inference of recursive functions: Complexity bounds. In J. Bārzdiņš and D. Bjørner, editors, *Baltic Computer Science*, volume 502 of *Lecture Notes in Computer Science*, pages 111–155. Springer-Verlag, 1991.

12 R. Freivalds, E. Kinber, and C. Smith. On the intrinsic complexity of learning. *Information and Computation*, 123(1):64–71, 1995.

13 E. M. Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

14 L. Hemaspaandra, H. Hempel, and G. Wechsung. Query order. *SIAM Journal of Computing*, 28:637–651, 1998.

15 S. Jain, Q. Luo, and F. Stephan. Learnability of automatic classes. In *Language and Automata Theory and Applications, 4th International Conference, LATA 2010*, volume 6031 of *LNCS*, pages 321–332. Springer, 2010.

16 S. Jain and A. Sharma. The structure of intrinsic complexity of learning. *Journal of Symbolic Logic*, 62:1187–1201, 1997.

17 B. Khoussainov and A. Nerode. Automatic presentations of structures. In *Logical and Computational Complexity, (International Workshop LCC 1994)*, volume 960 of *Lecture Notes in Computer Science*, pages 367–392. Springer, 1995.

18 S. Lange and T. Zeugmann. Language learning in dependence on the space of hypotheses. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, pages 127–136. ACM Press, 1993.

19 S. Lange and T. Zeugmann. Language learning with bounded number of mind changes. In *Proceedings of the Tenth Annual Symposium on Theoretical Aspects of Computer Science*, pages 682–691. Springer-Verlag, 1993. Lecture Notes Computer Science, 665.

20 D. Osherson, M. Stob, and S. Weinstein. *Systems that Learn: An Introduction to Learning Theory for Cognitive and Computer Scientists*. MIT Press, 1986.

21 H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967. Reprinted by MIT Press in 1987.

22 R. Wiehagen. On the complexity of effective program synthesis. In K. Jantke, editor, *Analogical and Inductive Inference, Proceedings of the International Workshop*, volume 265 of *Lecture Notes in Computer Science*, pages 209–219. Springer-Verlag, 1986.

23 T. Zeugmann and S. Zilles. Learning recursive functions: A survey. *Theoretical Computer Science A*, 397(1–3):4–56, 2008. Special Issue on Forty Years of Inductive Inference. Dedicated to the 60th Birthday of Rolf Wiehagen.

Monomials in arithmetic circuits: Complete problems in the counting hierarchy

Hervé Fournier[1], Guillaume Malod[1], and Stefan Mengel*[2]

1 **Univ Paris Diderot, Sorbonne Paris Cité,**
Institut de Mathématiques de Jussieu, UMR 7586 CNRS,
F-75205 Paris, France
{fournier,malod}@logique.jussieu.fr

2 **Institute of Mathematics**
University of Paderborn
D-33098 Paderborn, Germany
smengel@mail.uni-paderborn.de

Abstract

We consider the complexity of two questions on polynomials given by arithmetic circuits: testing whether a monomial is present and counting the number of monomials. We show that these problems are complete for subclasses of the counting hierarchy which had few or no known natural complete problems before. We also study these questions for circuits computing multilinear polynomials.

1998 ACM Subject Classification F.1.3 [Computation by abstract devices]: Complexity Measures and Classes — Reducibility and completeness

Keywords and phrases arithmetic circuits, counting problems, polynomials

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.362

1 Introduction

Several recent papers in arithmetic circuit complexity refer to a family of classes called the counting hierarchy consisting of the classes $\mathsf{PP} \cup \mathsf{PP}^{\mathsf{PP}} \cup \mathsf{PP}^{\mathsf{PP}^{\mathsf{PP}}} \cup \ldots$. For example, Bürgisser [6] uses these classes to connect computing integers to computing polynomials, while Jansen and Santhanam [14] — building on results by Koiran and Perifel [18] — use them to derive lower bounds from derandomization. This hierarchy was originally introduced by Wagner [32] to classify the complexity of combinatorial problems. Curiously, after Wagner's paper and another by Torán [27], this original motivation of the counting hierarchy has to the best of our knowledge not been pursued for more than twenty years. Instead, research focused on structural properties and the connection to threshold circuits [3]. As a result, there are very few natural complete problems for classes in the counting hierarchy: for instance, Kwisthout et al. give in [20] "the first problem with a practical application that is shown to be $\mathsf{FP}^{\mathsf{PP}^{\mathsf{PP}}}$-complete". The related class $\mathsf{C}_{=}\mathsf{P}$ appears to have no natural complete problems at all (see [13, p. 293]). It is however possible to define seemingly natural ones by starting with a #P-complete problem and considering the variant where an instance and a positive integer are provided and the question is to decide whether the number of solutions for this instance is equal to the integer. We consider these problems to be counting problems

* Partially supported by DFG grants BU 1371/2-2 and BU 1371/3-1.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 362–373

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

disguised as decision problems, in contrast to the question studied here. Note that the corresponding logspace counting class $\mathsf{C_{=}L}$ is known to have interesting complete problems from linear algebra [1].

In this paper we follow Wagner's original idea and show that the counting hierarchy is a helpful tool to classify the complexity of several natural problems on arithmetic circuits by showing complete problems for the classes $\mathsf{PP}^{\mathsf{PP}}$, $\mathsf{PP}^{\mathsf{NP}}$ and $\mathsf{C_{=}P}$.[1] The common setting of these problems is the use of circuits or straight-line programs to represent polynomials. Such a representation can be much more efficient than giving the list of monomials, but common operations on polynomials may become more difficult. An important example is the question of determining whether the given polynomial is identically zero. This is easy to do when given a list of monomials. When the polynomial is given as a circuit, the problem, called ACIT for *arithmetic circuit identity testing*, is solvable in coRP but is not known to be in P. In fact, derandomizing this problem would imply circuit lower bounds, as shown in [15]. This question thus plays a crucial part in complexity and it is natural to consider other problems on polynomials represented as circuits. In this article we consider mainly two questions.

The first question, called ZMC for *zero monomial coefficient*, is to decide whether a given monomial in a circuit has coefficient 0 or not. This problem has already been studied by Koiran and Perifel [17]. They showed that when the formal degree of the circuit is polynomially bounded the problem is complete for $\mathsf{P}^{\#\mathsf{P}}$. Unfortunately this result is not fully convincing, because it is formulated with the rather obscure notion of strong nondeterministic Turing reductions. We remedy this situation by proving a completeness result for the class $\mathsf{C_{=}P}$ under more traditional logarithmic space reductions. This provides a natural complete problem for this class. Koiran and Perifel also considered the general case of ZMC, where the formal degree of the circuits is not bounded. They showed that ZMC is in CH. We provide a better upper bound by proving that ZMC is in $\mathsf{coRP}^{\mathsf{PP}}$. We finally study the case of monotone circuits and show that the problem is then coNP-complete.

The second problem is to count the number of monomials in the polynomial computed by a circuit. This seems like a natural question whose solution should not be too hard, but in the general case it turns out to be $\mathsf{PP}^{\mathsf{PP}}$-complete, and the hardness holds even for weak circuits. We thus obtain another natural complete problem, in this case for the second level of the counting hierarchy.

Finally, we study the two above problems in the case of circuits computing multilinear polynomials. We show that our first problem becomes equivalent to the fundamental problem ACIT and that counting monomials becomes PP-complete.

2 Preliminaries

Complexity classes We assume the reader to be familiar with basic concepts of computational complexity theory (see e.g. [4]). All reductions in this paper will be logspace many-one unless stated otherwise.

We consider different counting decision classes in the counting hierarchy [32]. These classes are defined analogously to the quantifier definition of the polynomial hierarchy but, in addition to the quantifiers $\exists$ and $\forall$, the quantifiers C, $\mathsf{C_{=}}$ and $\mathsf{C_{\neq}}$ are used.

[1] Observe that Hemaspaandra and Ogihara [13, p. 293] state that Mundhenk et al. [24] provide natural complete problems for $\mathsf{PP}^{\mathsf{NP}}$. This appears to be a typo as Mundhenk et al. in fact present complete problems not for $\mathsf{PP}^{\mathsf{NP}}$ but for the class $\mathsf{NP}^{\mathsf{PP}}$ which indeed appears to have several interesting complete problems in the AI/planning literature.

▶ **Definition 2.1.** Let $\mathcal{C}$ be a complexity class.

- $A \in \mathsf{C}\mathcal{C}$ if and only if there is $B \in \mathcal{C}$, $f \in \mathsf{FP}$ and a polynomial p such that

$$x \in A \Leftrightarrow \left|\left\{y \in \{0,1\}^{p(|x|)} \mid (x,y) \in B\right\}\right| \geq f(x),$$

- $A \in \mathsf{C}_{=}\mathcal{C}$ if and only if there is $B \in \mathcal{C}$, $f \in \mathsf{FP}$ and a polynomial p such that

$$x \in A \Leftrightarrow \left|\left\{y \in \{0,1\}^{p(|x|)} \mid (x,y) \in B\right\}\right| = f(x),$$

- $A \in \mathsf{C}_{\neq}\mathcal{C}$ if and only if there is $B \in \mathcal{C}$, $f \in \mathsf{FP}$ and a polynomial p such that

$$x \in A \Leftrightarrow \left|\left\{y \in \{0,1\}^{p(|x|)} \mid (x,y) \in B\right\}\right| \neq f(x).$$

Observe that $\mathsf{C}_{\neq}\mathcal{C} = \mathsf{coC}_{=}\mathcal{C}$ with the usual definition $\mathsf{co}\mathcal{C} = \{L^c \mid L \in \mathcal{C}\}$, where L^c is the complement of L. That is why the quantifier $\mathsf{C}_{\neq}$ is often also written as $\mathsf{coC}_{=}$, so $\mathsf{C}_{\neq}\mathsf{P}$ is sometimes called $\mathsf{coC}_{=}\mathsf{P}$.

The counting hierarchy CH consists of the languages from all classes that we can get from P by applying the quantifiers $\exists$, $\forall$, C, $\mathsf{C}_{=}$ and $\mathsf{C}_{\neq}$ a constant number of times. Observe that with the definition above $\mathsf{PP} = \mathsf{CP}$. Torán [28] proved that this connection between PP and the counting hierarchy can be extended and that there is a characterization of CH by oracles similar to that of the polynomial hierarchy. We state some such characterizations which we will need later on, followed by other technical lemmas (we omit the proof of Lemma 2.3 which is not stated in [28] but can be shown with similar techniques).

▶ Lemma 2.2. [28] $\mathsf{PP}^{\mathsf{NP}} = \mathsf{C}\exists\mathsf{P}$.

▶ Lemma 2.3. $\mathsf{PP}^{\mathsf{PP}} = \mathsf{CC}_{\neq}\mathsf{P}$

▶ Lemma 2.4. [11] $\exists\mathsf{C}_{\neq}\mathsf{P} = \mathsf{C}_{\neq}\mathsf{P}$.

▶ Lemma 2.5. [25] For a large enough constant $c > 0$, it holds that for any integers n and x with $|x| \leqslant 2^{2^n}$ and $x \neq 0$, the number of primes p smaller than 2^{cn} such that $x \not\equiv 0 \mod p$ is at least $2^{cn}/cn$.

▶ Lemma 2.6. [13, p. 81] For every oracle X we have $\mathsf{PP}^{\mathsf{BPP}^X} = \mathsf{PP}^X$.

Arithmetic circuits An *arithmetic circuit* is a labeled directed acyclic graph (DAG) consisting of vertices or gates with indegree or fanin 0 or 2. The gates with fanin 0 are called input gates and are labeled with -1 or variables $X_1, X_2, \ldots, X_n$. The gates with fanin 2 are called computation gates and are labeled with $\times$ or $+$. We can also consider circuits where computation gates may receive more than two edges, in which case we say that they have *unbounded fanin*. The polynomial computed by an arithmetic circuit is defined in the obvious way: an input gate computes the value of its label, a computation gate computes the product or the sum of its children's values, respectively. We assume that a circuit has only one sink which we call the output gate. We say that the polynomial computed by the circuit is the polynomial computed by the output gate. The *size* of an arithmetic circuit is the number of gates. The *depth* of a circuit is the length of the longest path from an input gate to the output gate in the circuit. A formula is an arithmetic circuit whose underlying graph is a tree. Finally, a circuit or formula is called *monotone* if, instead of the constant -1, only the constant 1 is allowed.

It is common to consider so-called *degree-bounded* arithmetic circuits, for which the degree of the computed polynomial is bounded polynomially in the number of gates of the circuit.

In our opinion this kind of degree bound has two problems. One is that computing the degree of a polynomial represented by a circuit is suspected to be hard (see [2, 17, 16]), so problems defined with this degree bound must often be promise problems. The other problem is that the bound on the degree does not bound the size of computed constants, which by iterative squaring can have exponential bitsize. Thus even evaluating circuits on a Turing machine becomes intractable. The paper by Allender et al. [2] discusses problems that result from this. To avoid all these complications, instead of bounding the degree of the computed polynomial, we choose to bound the formal degree of the circuit or equivalently to consider multiplicatively disjoint circuits. A circuit is called *multiplicatively disjoint* if, for each $\times$-gate, its two input subcircuits are disjoint from one another. See [23] for a discussion of degree, formal degree and multiplicative disjointness and how they relate.

3 Zero monomial coefficient

We first consider the question of deciding if a single specified monomial occurs in a polynomial. In this problem and others regarding monomials, a monomial is encoded by giving the variable powers in binary.

ZMC
Input: Arithmetic circuit C, monomial m.
Problem: Decide if m has the coefficient 0 in the polynomial computed by C.

▶ Theorem 3.1. ZMC is $\mathsf{C}_=\mathsf{P}$-complete for both multiplicatively disjoint circuits and formulas.

Proof. Using standard reduction techniques from the #P-completeness of the permanent (see for example [4]), one define the following generic $\mathsf{C}_=\mathsf{P}$-complete problem, as mentioned in the introduction.

$\text{PER}_=$
Input: Matrix $A \in \{0, 1, -1\}^n$, $d \in \mathbb{N}$.
Problem: Decide if $\text{PER}(A) = d$.

Therefore, for the hardness of ZMC it is sufficient to show a reduction from $\text{PER}_=$. We use the following classical argument. On input $A = (a_{ij})$ and d we compute the formula $Q := \prod_{i=1}^{n} \left(\sum_{j=1}^{n} a_{ij} Y_j \right)$. It is a classical observation by Valiant [29][2] that the monomial $Y_1 Y_2 \dots Y_n$ has the coefficient $\text{PER}(A)$. Thus the coefficient of the monomial $Y_1 Y_2 \dots Y_n$ in $Q - dY_1 Y_2 \dots Y_n$ is 0 if and only if $\text{PER}(A) = d$.

We now show that ZMC for multiplicatively disjoint circuits is in $\mathsf{C}_=\mathsf{P}$. The proof is based on the use of parse trees, which can be seen as objects tracking the formation of monomials during the computation [23] and are the algebraic analog of proof trees [30]. A parse tree of a multiplicatively disjoint circuit is a subgraph with the following properties: it contains the output gate; if it contains a multiplication gate then it contains both its input edges; if it contains an addition gate then it contains exactly one of its input edges. The value of a parse tree is the product of the labels of all the input gates it contains. It is easy to see that the polynomial computed by a multiplicatively disjoint circuit is the sum of the values of all its parse trees.

[2] According to [31] this observation even goes back to [12].

Consider a multiplicatively disjoint circuit C and a monomial m, where the input gates of C are labeled either by a variable or by -1. A parse tree T contributes to the monomial m in the output polynomial if, when computing the value of the tree, we get exactly the powers in m; this contribution has coefficient $+1$ if the number of gates labeled -1 in T is even and it has coefficient -1 if this number is odd. The coefficient of m is thus equal to 0 if and only if the number of trees contributing positively is equal to the number of trees contributing negatively.

Let us represent a parse tree by a boolean word $\bar{\epsilon}$, by indicating which edges of C appear in the parse tree (the length N of the words is therefore the number of edges in C). Some of these words will not represent a valid parse tree, but this can be tested in polynomial time. Consider the following language L composed of triples $(C, m, \epsilon_0\bar{\epsilon})$ such that:

1. $\epsilon_0 = 0$ and $\bar{\epsilon}$ encodes a valid parse tree of C which contribute positively to m,
2. or $\epsilon_0 = 1$ and $\bar{\epsilon}$ does not encode a valid parse tree contributing negatively to m.

Then the number of $\bar{\epsilon}$ such that $(C, m, 0\bar{\epsilon})$ belongs to L is the number of parse trees contributing positively to m and the number of $\bar{\epsilon}$ such that $(C, m, 1\bar{\epsilon})$ belongs to L is equal to 2^N minus the number of parse trees contributing negatively to m. Thus, the number of $\epsilon_0\bar{\epsilon}$ such that $(C, m, \epsilon_0\bar{\epsilon}) \in L$ is equal to 2^N if and only if the number of trees contributing positively is equal to the number of trees contributing negatively, if and only if the coefficient of m is equal to 0 in C. Because L is in P, ZMC for multiplicatively disjoint circuits is in $\mathsf{C}_=\mathsf{P}$. ◀

▶ Theorem 3.2. ZMC belongs to $\mathsf{coRP}^{\mathsf{PP}}$.

Proof. Given a circuit C, a monomial m and a prime number p written in binary, CoeffSLP is the problem of computing modulo p the coefficient of the monomial m in the polynomial computed by C. It is shown in [16] (and implicitly in [22] and [17]) that CoeffSLP belongs to $\mathsf{FP}^{\#\mathsf{P}}$. See [9] for a more detailed proof simplifying the one in [22].

We now describe a randomized algorithm to decide ZMC. Let c be the constant given in Lemma 2.5. Consider the following algorithm to decide ZMC given a circuit C of size n and a monomial m, using CoeffSLP as an oracle. First choose uniformly at random an integer p smaller than 2^{cn}. If p is not prime, accept. Otherwise, compute the coefficient a of the monomial m in C with the help of the oracle and accept if $a \equiv 0 \mod p$. Since $|a| \leq 2^{2^n}$, Lemma 2.5 ensures that the above is a correct one-sided error probabilistic algorithm for ZMC. This yields ZMC $\in \mathsf{coRP}^{\text{CoeffSLP}}$. Hence ZMC $\in \mathsf{coRP}^{\mathsf{PP}}$. ◀

▶ Theorem 3.3. ZMC is coNP-complete both for monotone formulas and monotone circuits.

Proof. For hardness, we reduce the NP-complete problem Exact-3-Cover [10] to the complement of ZMC on monotone formulas, as done in [26, Chapter 3] (we reproduce the argument here for completeness).

Exact-3-Cover
Input: Integer n and $C_1, \ldots, C_m$ some 3-subsets of $\{1, \ldots, n\}$.
Problem: Decide if there exists $I \subseteq \{1, \ldots, m\}$ such that $\{C_i \mid i \in I\}$ is a partition of $\{1, \ldots, n\}$.

Consider the formula $F = \prod_{i=1}^{m}(1 + \prod_{j \in C_i} X_j)$. The monotone formula F has the monomial $\prod_{i=1}^{n} X_i$ if and only if $(n, C_1, \ldots, C_m)$ is a positive instance of Exact-3-Cover.

Let us now show that ZMC for monotone circuits is in coNP. This proof will use the notion of *parse tree types*, which are inspired by the generic polynomial introduced in [22]

to compute coefficient functions. We give here a sketch of the argument, more details are provided in [9]. The parse trees of a circuit which is not necessarily multiplicatively disjoint may be of a much bigger size than the circuit itself, because they can be seen as parse trees of the formula associated to the circuit and obtained by duplicating gates and edges. Define the *type* of a parse tree by giving, for each edge in the original circuit, the number of copies of this edge in the parse tree. There can be many different parse trees for a given parse tree type but they will all contribute to the same monomial, which is easy to obtain from the type: the power of a variable in the monomial is the sum, taken over all input gates labeled by this variable, of the number of edges leaving from this gate. In the case of a monotone circuit, computing the exact number of parse trees for a given type is thus not necessary, as a monomial will have a non-zero coefficient if and only if there exists a valid parse tree type producing this monomial.

Parse tree types, much like parse trees in the proof of Theorem 3.1, can be represented by Boolean tuples which must satisfy some easy-to-check conditions to be valid. Thus the coefficient of a monomial is 0 if and only if there are no valid parse tree types producing this monomial, which is a coNP condition. ◀

4 Counting monomials

We now turn to the problem of counting the monomials of a polynomial represented by a circuit.

CountMon
Input: Arithmetic circuit C, $d \in \mathbb{N}$.
Problem: Decide if the polynomial computed by C has at least d monomials.

To study the complexity of CountMon we will look at what we call extending polynomials. Given two monomials M and m, we say that M is m-extending if $M = mm'$ and m and m' have no common variable. We start by studying the problem of deciding the existence of an extending monomial.

ExistExtMon
Input: Arithmetic circuit C, monomial m.
Problem: Decide if the polynomial computed by C contains an m-extending monomial.

▶ Proposition 4.1. ExistExtMon is in $\mathsf{RP}^{\mathsf{PP}}$. For multiplicatively disjoint circuits it is $\mathsf{C}_{\neq}\mathsf{P}$-complete.

Proof. We first show the first upper bound. So let (C, m) be an input for ExistExtMon where C is a circuit in the variables $X_1, \ldots, X_n$. Without loss of generality, suppose that $X_1, \ldots, X_r$ are the variables appearing in m. Let $d = 2^{|C|}$: d is a bound on the degree of the polynomial computed by C. We define $C' = \prod_{i=r+1}^{n}(1 + Y_i X_i)^d$ for new variables Y_i. We have that C has an m-extending monomial if and only if in the product CC' the polynomial $P(Y_{r+1}, \ldots, Y_n)$, which is the coefficient of $m \prod_{i=r+1}^{n} X_i^d$, is not identically 0. Observe that P is not given explicitly but can be evaluated modulo a random prime with an oracle for CoeffSLP. Thus it can be checked if P is identically 0 with the classical Schwartz-Zippel-DeMillo-Lipton lemma (see for example [4]). It follows that ExistExtMon $\in \mathsf{RP}^{\mathsf{PP}}$.

The upper bound in the multiplicatively disjoint setting is easier: we can guess an m-extending monomial M and then output the answer of an oracle for the complement of ZMC,

to check whether M appears in the computed polynomial. This establishes containment in $\exists\mathsf{C}_{\neq}\mathsf{P}$ which by Lemma 2.4 is $\mathsf{C}_{\neq}\mathsf{P}$.

For hardness we reduce to ExistExtMon the $\mathsf{C}_{\neq}\mathsf{P}$-complete problem $\text{PER}_{\neq}$, i.e., the complement of the $\text{PER}_{=}$ problem introduced for the proof of Theorem 3.1. We use essentially the same reduction constructing a circuit $Q := \prod_{i=1}^{n}\left(\sum_{j=1}^{n} a_{ij}Y_j\right)$. Observe that the only potential extension of $m := Y_1Y_2\dots Y_n$ is m itself and has the coefficient $\text{PER}(A)$. Thus $Q - dY_1Y_2\dots Y_n$ has an m-extension if and only if $\text{PER}(A) \neq d$. ◀

CountExtMon
Input: Arithmetic circuit C, $d \in \mathbb{N}$, monomial m.
Problem: Decide if the polynomial computed by C has at least d m-extending monomials.

▶ Proposition 4.2. CountExtMon is $\mathsf{PP}^{\mathsf{PP}}$-complete.

Proof. Clearly CountExtMon belongs to PP^{ZMC} and thus with Theorem 3.2 it is in $\mathsf{PP}^{\mathsf{coRP}^{\mathsf{PP}}}$. Using Lemma 2.6 we get membership in $\mathsf{PP}^{\mathsf{PP}}$. To show hardness, we reduce the canonical $\mathsf{CC}_{\neq}\mathsf{P}$-complete problem $\mathsf{CC}_{\neq}$3SAT to CountExtMon. With Lemma 2.3 the hardness for $\mathsf{PP}^{\mathsf{PP}}$ follows.

$\mathsf{CC}_{\neq}$3SAT
Input: 3SAT-formula $F(\bar{x},\bar{y})$, $k,\ell \in \mathbb{N}$.
Problem: Decide if there are at least k assignments to $\bar{x}$ such that there are not exactly ℓ assignments to $\bar{y}$ such that F is satisfied.

Let $(F(\bar{x},\bar{y}),k,\ell)$ be an instance for $\mathsf{CC}_{\neq}$3SAT. Without loss of generality we may assume that $\bar{x} = (x_1,\dots,x_n)$ and $\bar{y} = (y_1,\dots,y_n)$ and that no clause contains a variable in both negated and unnegated form. Let $\Gamma_1,\dots,\Gamma_c$ be the clauses of F.

For each literal u of the variables in $\bar{x}$ and $\bar{y}$ we define a monomial $I(u)$ in the variables $X_1,\dots,X_n,Z_1,\dots,Z_c$ in the following way:

$$I(x_i) = X_i \prod_{\{j \mid x_i \in \Gamma_j\}} Z_j \qquad I(\neg x_i) = \prod_{\{j \mid \neg x_i \in \Gamma_j\}} Z_j$$

$$I(y_i) = \prod_{\{j \mid y_i \in \Gamma_j\}} Z_j \qquad I(\neg y_i) = \prod_{\{j \mid \neg y_i \in \Gamma_j\}} Z_j$$

From these monomials we compute a formula C by

$$C := \prod_{i=1}^{n} (I(x_i) + I(\neg x_i)) \prod_{i=1}^{n} (I(y_i) + I(\neg y_i)). \tag{1}$$

We fix a mapping mon from the assignments of F to the monomials computed by C: Let $\bar{\alpha}$ be an assignment to $\bar{x}$ and $\bar{\beta}$ be an assignment to $\bar{y}$. We define $mon(\bar{\alpha}\bar{\beta})$ as the monomial obtained in the expansion of C by choosing the following terms. If $\alpha_i = 0$, choose $I(\neg x_i)$, otherwise choose $I(x_i)$. Similarly, if $\beta_i = 0$, choose $I(\neg y_i)$, otherwise choose $I(y_i)$.

The monomial $mon(\bar{\alpha}\bar{\beta})$ has the form $\prod_{i=1}^{n} X_i^{\alpha_i} \prod_{j=1}^{c} Z_j^{\gamma_j}$, where γ_j is the number of true literals in Γ_j under the assignment $\bar{\alpha}\bar{\beta}$. Then F is true under $\bar{\alpha}\bar{\beta}$ if and only if $mon(\bar{\alpha}\bar{\beta})$ has the factor $\prod_{j=1}^{c} Z_j$. Thus F is true under $\bar{\alpha}\bar{\beta}$ if and only if $mon(\bar{\alpha}\bar{\beta}) \prod_{j=1}^{c}\left(1 + Z_j + Z_j^2\right)$ has the factor $\prod_{i=1}^{n} X_i^{\alpha_i} \prod_{j=1}^{c} Z_j^3$. We set $C' = C \prod_{j=1}^{c}\left(1 + Z_j + Z_j^2\right)$.

Consider an assignment $\bar{\alpha}$ to $\bar{x}$. The coefficient of the monomial $\prod_{i=1}^{n} X_i^{\alpha_i} \prod_{j=1}^{c} Z_j^3$ in C' is the number of assignments $\bar{\beta}$ such that $\bar{\alpha}\bar{\beta}$ satisfies F. Thus we get

$$
\begin{aligned}
& (F(\bar{x},\bar{y}),k,\ell) \in \mathsf{CC}_{\neq}\mathrm{3SAT} \\
\Leftrightarrow\ & \text{there are at least } k \text{ assignments } \bar{\alpha} \text{ to } \bar{x} \text{ such that the monomial } \prod_{i=1}^{n} X_i^{\alpha_i} \prod_{j=1}^{c} Z_j^3 \\
& \text{does not have coefficient } \ell \text{ in } C' \\
\Leftrightarrow\ & \text{there are at least } k \text{ assignments } \bar{\alpha} \text{ to } \bar{x} \text{ such that the monomial } \prod_{i=1}^{n} X_i^{\alpha_i} \prod_{j=1}^{c} Z_j^3 \\
& \text{occurs in } C'' := C' - \ell \prod_{i=1}^{n}(1+X_i) \prod_{j=1}^{c} Z_j^3 \\
\Leftrightarrow\ & \text{there are at least } k \text{ tuples } \bar{\alpha} \text{ such that } C'' \text{ contains the monomial } \prod_{i=1}^{n} X_i^{\alpha_i} \prod_{j=1}^{c} Z_j^3 \\
\Leftrightarrow\ & C'' \text{ has at least } k \ (\prod_{j=1}^{c} Z_j^3)\text{-extending monomials.}
\end{aligned}
$$

◀

▶ Theorem 4.3. CountMon is $\mathsf{PP}^{\mathsf{PP}}$-complete. It is $\mathsf{PP}^{\mathsf{PP}}$-hard even for unbounded fan-in formulas of depth 4.

Proof. CountMon can be easily reduced to CountExtMon since the number of monomials of a polynomial is the number of 1-extending monomials. Therefore CountMon belongs to $\mathsf{PP}^{\mathsf{PP}}$.

To show hardness, it is enough to prove that instances of CountExtMon constructed in Proposition 4.2 can be reduced to CountMon in logarithmic space. The idea of the proof is that we make sure that the polynomial for which we count all monomials contains all monomials that are not m-extending. Thus we know how many non-m-extending monomials it contains and we can compute the number of m-extending monomials from the number of all monomials. We could use the same strategy to show in general that CountExtMon reduces to CountMon but by considering the instance obtained in the proof of Proposition 4.2 and analyzing the extra calculations below we get hardness for unbounded fanin formulas of depth 4.

So let (C'', k, m) be the instance of CountExtMon constructed in the proof of Proposition 4.2, with $m = \prod_{j=1}^{c} Z_j^3$. We therefore need to count the monomials computed by C'' which are of the form $f(X_1,\ldots,X_n)\prod_{j=1}^{c} Z_j^3$. The circuit C'' is multilinear in X, and the Z_j can only appear with powers in $\{0,1,2,3,4,5\}$. So the non-m-extending monomials computed by C'' are all products of a multilinear monomial in the X_i and a monomial in the Z_j where at least one Z_j has a power in $\{0,1,2,4,5\}$. Fix j, then all monomials that are not m-extending because of Z_j are computed by the formula

$$\tilde{C_j} := \left(\prod_{i=1}^{n}(X_i+1)\right)\left(\prod_{j'\neq j}\sum_{p=0}^{5} Z_{j'}^p\right)\left(1+Z_j+Z_j^2+Z_j^4+Z_j^5\right). \tag{2}$$

Thus the formula $\tilde{C} := \sum_j \tilde{C_j}$ computes all non-m-extending monomials that C'' can compute. The coefficients of monomials in C'' cannot be smaller than $-\ell$ where ℓ is part of the instance of $\mathsf{CC}_{\neq}$3SAT from which we constructed (C'', k, m) before. So the formula

$C^* := C'' + (\ell+1)\tilde{C}$ contains all non-m-extending monomials that C'' can compute and it contains the same extending monomials. There are $2^n 6^c$ monomials of the form that C'' can compute, only 2^n of which are m-extending, which means that there are $2^n(6^c-1)$ monomials computed by C^* that are not m-extending. As a consequence, C'' has at least k m-extending monomials if and only if C^* has at least $2^n(6^c-1)+k$ monomials. ◀

▶ Theorem 4.4. CountMon is $\mathsf{PP}^{\mathsf{NP}}$-complete both for monotone formulas and monotone circuits.

Proof. We first show hardness for monotone formulas. The argument is very similar to the proof of Theorem 4.3. Consider the following canonical $\mathsf{C}\exists\mathsf{P}$-complete problem $\mathsf{C}\exists$3SAT.

$\mathsf{C}\exists$3SAT
Input: 3SAT-formula $F(\bar{x},\bar{y})$, $k \in \mathbb{N}$.
Problem: Decide if there are at least k assignments $\bar{\alpha}$ to $\bar{x}$ such that $F(\bar{\alpha},\bar{y})$ is satisfiable.

We reduce $\mathsf{C}\exists$3SAT to CountMon. With Lemma 2.2 the hardness for $\mathsf{PP}^{\mathsf{NP}}$ follows. Consider a 3SAT-formula $F(\bar{x},\bar{y})$. Let $n = |\bar{x}| = |\bar{y}|$ and let c be the number of clauses of F. Define the polynomial $C^* = C + \sum_{j=1}^{c} \tilde{C}_j$ where C is defined by Equation 1 and $\tilde{C}_j$ by Equation 2. The analysis is similar to the proof of Theorem 4.3. The polynomial C^* is computed by a monotone arithmetic formula and has at least $2^n(6^c-1)+k$ monomials if and only if (F,k) is a positive instance of $\mathsf{C}\exists$3SAT.

We now prove the upper bound. Recall that CountMon $\in \mathsf{PP}^{\mathrm{ZMC}}$. From Theorem 3.3, it follows that CountMon on monotone circuits belongs to $\mathsf{PP}^{\mathsf{NP}}$. ◀

5 Multilinearity

In this section we consider the effect of multilinearity on our problems. We will not consider promise problems and therefore the multilinear variants of our problems must first check if the computed polynomial is multilinear. We start by showing that this step is not difficult. The proof is omitted due to space constraints.

CheckML
Input: Arithmetic circuit C.
Problem: Decide if the polynomial computed by C is multilinear.

▶ Proposition 5.1. CheckML is equivalent to ACIT.

Next we show that the problem gets much harder if, instead of asking whether *all* the monomials in the polynomial computed by a circuit are multilinear, we ask whether at least *one* of the monomials is multilinear.

MonML
Input: Arithmetic circuit C.
Problem: Decide if the polynomial computed by C contains a multilinear monomial.

The problem monML lies at the heart of fast exact algorithms for deciding k-paths by Koutis and Williams [19, 33] (although in these papers the polynomials are in characteristic 2 which changes the problem a little). This motivated Chen and Fu [7, 8] to consider monML, show that it is #P-hard and give algorithms for the bounded depth version. We

provide further information on the complexity of this problem (the proof is similar to that of Proposition 4.1 and can be found in [9]).

▶ Proposition 5.2. MonML is in $\mathsf{RP}^{\mathsf{PP}}$. It is $\mathsf{C}_{\neq}\mathsf{P}$-complete for multiplicatively disjoint circuits.

We now turn to our first problem, namely deciding whether a monomial appears in the polynomial computed by a circuit, in the multilinear setting.

ML-ZMC
Input: Arithmetic circuit C, monomial m.
Problem: Decide if C computes a multilinear polynomial in which the monomial m has coefficient 0.

▶ Proposition 5.3. ML-ZMC is equivalent to ACIT.

Proof. We first show that ACIT reduces to ML-ZMC. So let C be an input for ACIT. Allender et al. [2] have shown that ACIT reduces to a restricted version of ACIT in which all inputs are -1 and thus the circuit computes a constant. Let C_1 be the result of this reduction. Then C computes identically 0 if and only if the constant coefficient of C_1 is 0. This establishes the first direction.

For the other direction let (C, m) be the input, where C is an arithmetic circuit and m is a monomial. First check if m is multilinear, if not output 1 or any other nonzero polynomial. Next we construct a circuit C_1 that computes the homogeneous component of degree $\deg(m)$ of C with the classical method (see for example [5, Lemma 2.14]). Observe that if C computes a multilinear polynomial, so does C_1. We now plug in 1 for the variables that appear in m and 0 for all other variables, call the resulting (constant) circuit C_2. If C_1 computes a multilinear polynomial, then C_2 is zero if and only if m has coefficient 0 in C_1. The end result of the reduction is $C^* := C_2 + ZC_3$ where Z is a new variable and C_3 is a circuit which is identically 0 iff C computes a multilinear polynomial (obtained via Proposition 5.1). C computes a multilinear polynomial and does not contain the monomial m if and only if both C_2 and ZC_3 are identically 0, which happens if and only if their sum is identically 0. ◀

In the case of our second problem, counting the number of monomials, the complexity falls to PP.

ML-CountMon
Input: Arithmetic circuit C, $d \in \mathbb{N}$.
Problem: Decide if the polynomial computed by C is multilinear and has at least d monomials.

▶ Proposition 5.4. ML-CountMon is PP-complete (for Turing reductions).

Proof. We first show ML-CountMon $\in$ PP. To do so we use CheckML to check that the polynomial computed by C is multilinear. Then counting monomials can be done in $\mathsf{PP}^{\text{ML-ZMC}}$, and ML-ZMC is in coRP. By Lemma 2.6 the class $\mathsf{PP}^{\mathsf{coRP}}$ is simply PP.

For hardness we reduce the computation of the $\{0,1\}$-permanent to ML-CountMon. The proposition follows, because the $\{0,1\}$-permanent is #P-complete for Turing reductions. So let A be a 0-1-matrix and $d \in \mathbb{N}$ and we have to decide if $\mathrm{per}(A) \geq d$. We get a matrix B from A by setting $b_{ij} := a_{ij}X_{ij}$. Because every entry of B is either 0 or a distinct variable, we have that, when we compute the permanent of B, every permutation that yields a non-zero

summand yields a unique monomial. This means that there are no cancellations, so that $\text{PER}(A)$ is the number of monomials in $\text{PER}(B)$.

The problem is now that no small circuits for the permanent are known and thus $\text{PER}(B)$ is not a good input for ML-CountMon. But because there are no cancellations, we have that $\text{DET}(B)$ and $\text{PER}(B)$ have the same number of monomials. So take a small circuit for the determinant (for instance the one given in [21]) and substitute its inputs by the entries of B. The result is a circuit C which computes a polynomial whose number of monomials is $\text{PER}(A)$. Observing that the determinant, and thus the polynomial computed by C, is multilinear completes the proof. ◀

Acknowledgements We would like to thank Sylvain Perifel for helpful discussions. The results of this paper were conceived while the third author was visiting the Équipe de Logique Mathématique at Université Paris Diderot Paris 7. He would like to thank Arnaud Durand for making this stay possible, thanks to funding from ANR ENUM (ANR-07-BLAN-0327). The third author would also like to thank his supervisor Peter Bürgisser for helpful advice.

References

1 E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8(2):99–126, 1999.

2 E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the Complexity of Numerical Analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.

3 E.W. Allender and K.W. Wagner. Counting hierarchies: polynomial time and constant depth circuits. *Current trends in theoretical computer science: essays and Tutorials*, 40:469, 1993.

4 S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

5 P. Bürgisser. *Completeness and reduction in algebraic complexity theory*, volume 7. Springer Verlag, 2000.

6 P. Bürgisser. On Defining Integers And Proving Arithmetic Circuit Lower Bounds. *Computational Complexity*, 18(1):81–103, 2009.

7 Zhixiang Chen and Bin Fu. Approximating multilinear monomial coefficients and maximum multilinear monomials in multivariate polynomials. In *COCOA (1)*, pages 309–323, 2010.

8 Zhixiang Chen and Bin Fu. The Complexity of Testing Monomials in Multivariate Polynomials. In *COCOA*, pages 1–15, 2011.

9 H. Fournier, G. Malod, and S. Mengel. Monomials in arithmetic circuits: Complete problems in the counting hierarchy. *ArXiv e-prints*, October 2011.

10 M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

11 F. Green. On the Power of Deterministic Reductions to $C_{=}P$. *Theory of Computing Systems*, 26(2):215–233, 1993.

12 J. Hammond. Question 6001. *Educ. Times*, 32:179, 1879.

13 L.A. Hemaspaandra and M. Ogihara. *The complexity theory companion*. Springer Verlag, 2002.

14 M. Jansen and R. Santhanam. Permanent Does Not Have Succinct Polynomial Size Arithmetic Circuits of Constant Depth. In *ICALP*, pages 724–735, 2011.

15 V. Kabanets and R. Impagliazzo. Derandomizing Polynomial Identity Tests Means Proving Circuit Lower Bounds. *Computational Complexity*, 13:1–46, 2004.

16 N. Kayal and C. Saha. On the Sum of Square Roots of Polynomials and related problems. In *IEEE Conference on Computational Complexity*, 2011.

17 P. Koiran and S. Perifel. The complexity of two problems on arithmetic circuits. *Theor. Comput. Sci.*, 389(1-2):172–181, 2007.

18 P. Koiran and S. Perifel. Interpolation in Valiant's Theory. *Computational Complexity*, pages 1–20, 2011.

19 I. Koutis. Faster Algebraic Algorithms for Path and Packing Problems. In *ICALP*, pages 575–586, 2008.

20 J. H. P. Kwisthout, H. L. Bodlaender, and L. C. Van Der Gaag. The complexity of finding kth most probable explanations in probabilistic networks. In *Proceedings of the 37th international conference on Current trends in theory and practice of computer science*, SOFSEM'11, pages 356–367, Berlin, Heidelberg, 2011. Springer-Verlag.

21 M. Mahajan and V. Vinay. A combinatorial algorithm for the determinant. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 730–738. Society for Industrial and Applied Mathematics, 1997.

22 G. Malod. The Complexity of Polynomials and Their Coefficient Functions. In *IEEE Conference on Computational Complexity*, pages 193–204, 2007.

23 G. Malod and N. Portier. Characterizing Valiant's algebraic complexity classes. *J. Complexity*, 24(1):16–38, 2008.

24 M. Mundhenk, J. Goldsmith, C. Lusena, and E. Allender. Complexity of Finite-Horizon Markov Decision Process Problems. *Journal of the ACM (JACM)*, 47(4):681–720, 2000.

25 A. Schönhage. On the Power of Random Access Machines. In *ICALP*, pages 520–529, 1979.

26 Y. Strozecki. *Enumeration complexity and matroid decomposition*. PhD thesis, Université Paris Diderot - Paris 7, 2010.

27 J. Torán. Succinct Representations of Counting Problems. In *AAECC*, pages 415–426, 1988.

28 J. Torán. Complexity Classes Defined by Counting Quantifiers. *J. ACM*, 38(3):753–774, 1991.

29 L.G. Valiant. Completeness classes in algebra. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 249–261. ACM, 1979.

30 H. Venkateswaran and M. Tompa. A New Pebble Game that Characterizes Parallel Complexity Classes. *SIAM J. Comput.*, 18(3):533–549, 1989.

31 J. Von Zur Gathen. Feasible arithmetic computations: Valiant's hypothesis. *Journal of Symbolic Computation*, 4(2):137–172, 1987.

32 K. W. Wagner. The Complexity of Combinatorial Problems with Succinct Input Representation. *Acta Informatica*, 23(3):325–356, 1986.

33 R. Williams. Finding paths of length k in $O^*(2^k)$ time. *Information Processing Letters*, 109(6):315–318, 2009.

Motion planning with pulley, rope, and baskets*

Christian E.J. Eggermont[1] and Gerhard J. Woeginger[1]

1 **Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, Netherlands**

Abstract

We study a motion planning problem where items have to be transported from the top room of a tower to the bottom of the tower, while simultaneously other items have to be transported into the opposite direction. Item sets are moved in two baskets hanging on a rope and pulley. To guarantee stability of the system, the weight difference between the contents of the two baskets must always stay below a given threshold.

We prove that it is Π_2^p-complete to decide whether some given initial situation of the underlying discrete system can lead to a given goal situation. Furthermore we identify several polynomially solvable special cases of this reachability problem, and we also settle the computational complexity of a number of related questions.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases planning and scheduling; computational complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.374

1 Introduction

The Oxford mathematician Charles Lutwidge Dodgson (1832–1898) is better known under his pseudonym Lewis Carroll. He is the author of books like *"Alice's Adventures in Wonderland"* and *"Through the Looking-Glass"*, and he has constructed a multitude of mathematical puzzles. One of Carroll's most famous problems is called *"The Captive Queen"*; see for instance Wakeling [10]:

> *"A captive queen and her son and daughter were shut up in the top room of a very high tower. Outside their window was a pulley with a rope around it, and a basket fastened to each end of the rope of equal weight. They managed to escape with the help of this and a weight they found in the room, quite safely. It would have been dangerous for any of them to come down if they weighed 15 lbs more than the content of the other basket, for they would do so too quick, and they also managed not to weigh less either. The one basket coming down would naturally of course draw the other basket up.*
> *The queen weighed 195 lbs, daughter 105, son 90, and the weight 75 lbs. How did they all escape safely?"*

In the initial situation queen, daughter, son, and weight are all up the tower and none of them is at the bottom of the tower. This situation is denoted $Q, D, S, W \parallel \emptyset$, and we use a similarly intuitive notation for other situations. The schedule in Figure 1 solves the Captive Queen problem in eleven steps.

* This research has been supported by the Netherlands Organisation for Scientific Research (NWO), grant 639.033.403, and by DIAMANT (an NWO mathematics cluster)

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 374–383

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

1.	The weight is sent down	$Q, D, S \parallel W$
2.	The son goes down, the weight comes up	$Q, D, W \parallel S$
3.	The daughter goes down, and the son comes up	$Q, S, W \parallel D$
4.	The weight goes down	$Q, S \parallel D, W$
5.	The queen goes down; daughter and weight come up	$D, S, W \parallel Q$
6.	The weight falls down	$D, S \parallel Q, W$
7.	The son goes down, the weight comes up	$D, W \parallel Q, S$
8.	The daughter goes down, and the son comes up	$S, W \parallel Q, D$
9.	The son sends down the weight	$S \parallel Q, D, W$
10.	The son goes down, the weight comes up	$W \parallel Q, D, S$
11.	The weight falls to the ground	$\emptyset \parallel Q, D, S, W$

Figure 1 A feasible schedule for Lewis Carroll's Captive Queen problem.

Mathematical formulation

Motivated by the Captive Queen problem, we will investigate the following motion planning problem. Let I be a set of items, and let $w(i)$ be the positive integer weight of item $i \in I$. A *state* of the system is specified by an item set $J \subseteq I$ at the top of the tower (and with the remaining items in $I - J$ located at the bottom of the tower). For $J \subseteq I$ we throughout denote $w(J) = \sum_{j \in J} w(j)$, and as usual we let $w(\emptyset) = 0$. The system can move directly from state $J \subseteq I$ to state $K \subseteq I$ if

$$|w(J \cap (I - K)) - w(K \cap (I - J))| \ \leq \ \Delta, \tag{1}$$

where the positive integer bound Δ specifies the maximum allowed weight difference between the two exchanged subsets in the baskets. We say that state K is *reachable* from state J, if there is a sequence of moves that transforms J into K. Since inequality (1) is symmetric in J and K, reachability is a symmetric relation. The decision version of our mathematical motion planning problem is defined as follows.

Problem: Captive-Queen

Instance: A set I of items; positive integer weights $w(i)$ for $i \in I$; a positive integer bound Δ; two subsets $I_0, I_1 \subseteq I$.

Question: Is the goal state I_1 reachable from the initial state I_0?

Although problem Captive-Queen does not cover all the algorithmic features of Carroll's problem, we think that it does cover the most important ones. Note that the weight of 75 lbs in Carroll's problem constitutes an indestructible item that can fall down with the basket without obeying constraint (1); in our problem formulation, however, there are no such indestructible items.

We will also discuss the following variant of Captive-Queen where the number of moves is a priori bounded by m.

Problem: Captive-Queen-with-few-Moves
Instance: A set I of items; integer weights $w(i)$ for $i \in I$; a bound Δ; two subsets $I_0, I_1 \subseteq I$; an integer bound m that is encoded in unary.
Question: Is there a sequence of at most m moves that transforms the initial state I_0 into the goal state I_1?

The following example illustrates that there exist YES-instances of Captive-Queen for which every feasible schedule has exponential length.

▶ **Example 1.** Consider the item set $I = \{0, 1, \ldots, n-1\}$ with weights $w(i) = 2^i$ for $i \in I$. The difference bound is $\Delta = 1$, the initial state is $I_0 = \emptyset$, and the goal state is $I_1 = I$.

Let $J_1, \ldots, J_{2^n}$ be an enumeration of all 2^n subsets of I in order of increasing weight. By considering the binary representation of $w(J_j)$ and $w(J_{j+1})$, one sees that the system can move from every state J_j to the successor state J_{j+1}. Since $J_1 = I_0$ and $J_{2^n} = I_1$, there consequently exists a sequence of $2^n - 1$ moves that transforms state I_0 into state I_1. Since every move increases the weight of the current state by at most $\Delta = 1$, every feasible schedule must have length at least $2^n - 1$.

Our results

We establish a number of results on the algorithmic and combinatorial behavior of the motion planning problems introduced above. As our main result, we precisely pinpoint the computational complexity of the Captive-Queen problem: it is Π_2^p-complete and hence located at the second level of the polynomial hierarchy (Section 3). The variant Captive-Queen-with-few-Moves turns out to be NP-complete. Next we show that certain natural special cases of Captive-Queen are polynomially solvable:

- the case with super-increasing weight sequences (Section 4.1);
- the case with divisible weight sequences (Section 4.2).

These special cases originate from the literature around the knapsack problem (see for instance the books [6, 4]). In Section 5 we finally characterize the computational complexity of several related algorithmic problems:

- recognizing isolated states in a system;
- deciding whether every state in a system is isolated;
- deciding whether a system contains some isolated state;
- deciding whether all states in a system are reachable from each other.

We also discuss the restriction of these problems to super-increasing weight sequences and to divisible weight sequences.

2 Preliminaries and first observations

We consider some fixed instance of Captive-Queen with item set I, weights $w(i)$, difference bound Δ, and initial state I_0 and goal state I_1. Throughout the paper we will assume without loss of generality that $w(I_0) \leq w(I_1)$ (and otherwise we simply swap I_0 and I_1).

The following (straightforward) lemma provides a concise characterization of the possible moves between states.

▶ **Lemma 2.** *There is a direct move from state $J \subseteq I$ to state $K \subseteq I$ if and only if*

$$|w(J) - w(K)| \;\leq\; \Delta. \tag{2}$$

Proof. This follows since the inequalities in (1) and (2) are equivalent. ◀

The *weight spectrum* $W_1 < W_2 < \cdots < W_k$ of the Captive-Queen instance enumerates the weights of all the subsets of I in increasing order. The *weight spectrum between* $w(I_0)$ *and* $w(I_1)$ is the piece $W_a < \cdots < W_b$ of the weight spectrum starting with $W_a = w(I_0)$ and ending with $W_b = w(I_1)$. The *maximum gap* between two bounds W_a and W_b is the maximum of the values $W_{j+1} - W_j$ taken over $j = a, \ldots, b-1$.

The standard dynamic programming algorithm for the Subset-Sum problem generates (as a by-product) a sorted list of the sums of all subsets of a given set of integers; see for instance Cormen & al [2]. This yields the following.

▶ **Lemma 3.** *The weight spectrum can be computed in pseudo-polynomial time* $O(nW)$*, where* $n = |I|$ *and* $W = \sum_{i \in I} w(i)$*.* ◀

The following observation is an immediate consequence of Lemma 2.

▶ **Corollary 4.** *The following three statements are pairwise equivalent.*

(i) *The goal state* I_1 *is reachable from the initial state* I_0*.*
(ii) *The maximum gap in the weight spectrum between* $w(I_0)$ *and* $w(I_1)$ *is at most* Δ*.*
(iii) *For every integer* V *with* $w(I_0) \leq V < w(I_1)$*, there exists an item set* $J \subseteq I$ *such that* $V < w(J) \leq V + \Delta$*.*

Lemma 3 and Corollary 4.(ii) together imply that Captive-Queen and Captive-Queen-with-few-Moves are solvable in pseudo-polynomial time.

3 Hardness of the Captive-Queen

In this section we will establish Captive-Queen to be Π_2^p-complete and Captive-Queen-with-few-Moves to be NP-complete.

Corollary 4.(iii) shows that the Captive-Queen problem can be rewritten into an equivalent question of the form $\forall x \exists y\, P(x, y)$ where $P(x, y)$ is a Boolean predicate that can be evaluated in polynomial time. The complexity class Π_2^p represents problems of exactly this particular form with a universal quantifier followed by an existential quantifier (see for instance Section 17.2 in Papadimitriou [7]). Hence we derive the following statement.

▶ **Lemma 5.** *Problem* Captive-Queen *lies in* Π_2^p*.* ◀

The main part of this section is dedicated to proving Π_2^p-hardness of the Captive-Queen problem. The proof is done by means of a polynomial time reduction from the following quantified satisfiability problem, which was shown to be Π_2^p-complete by Stockmeyer [9].

Problem: 2-Quantified 3-CNF-Sat

Instance: Two sets $X = \{x_1, \ldots, x_s\}$ and $Y = \{y_1, \ldots, y_s\}$ of Boolean variables. A Boolean formula $\phi(X, Y)$ over $X \cup Y$ in conjunctive normal with clauses $c_1, \ldots, c_t$ where every (disjunctive) clause c_j consists of exactly three literals.

Question: Is $\forall x_1, \ldots, x_s\, \exists y_1, \ldots, y_s\ \phi(X, Y)$ true?

We pick an arbitrary instance of 2-Quantified 3-CNF-Sat, and we will construct a corresponding instance of Captive-Queen from it. In our construction every item weight is specified in terms of its decimal representation, which consists of $3s + 2t$ digits that are partitioned into five parts; see Figure 2 for an illustration.

- The verification-part consists of the t left-most digits in the decimal representation, and the clause-part consists of the t digits immediately to the right of the verification-part. In both parts the jth digit from the right ($1 \leq j \leq t$) is said to correspond to clause c_j.
- The Y-part consists of the next s digits. In this part the ith digit from the right ($1 \leq i \leq s$) corresponds to variable y_i.
- The X-part consists of the next s digits (immediately to the right of the Y-part). The ith digit from the right ($1 \leq i \leq s$) corresponds to the Boolean variable x_i.
- The control-part consists of the remaining s digits in the decimal representation. For technical reasons, we will mainly work with the s lowest bits in the *binary* representation of the control-part (and we will ignore the remaining unused bits). The ith bit from the right ($1 \leq i \leq s$) corresponds to the Boolean variable x_i.

Verification part	**Clause-part**	**Y-part**	**X-part**	**Control part**
$t \ldots\ldots 1$	$t \ldots\ldots 1$	$s \ldots\ldots 1$	$s \ldots\ldots 1$	$s \ldots\ldots 1$

Figure 2 The division of the decimal representations into five parts.

Throughout we will use the term *digits* to specify the decimal representation of the verification-part, clause-part, Y-part, and X-part, and we will use the term *bits* to specify the binary representation of the control-part. Next, let us describe the $4s + 3t + 2$ items in the Captive-Queen instance together with their weights.

- For every literal $\ell \in \{x_i, \overline{x_i}\}$ there is a corresponding X-item $X(\ell)$. The weight of $X(\ell)$ has a 1-digit in the position corresponding to variable x_i in the X-part. If $\ell = x_i$ is unnegated, then there is a 1-bit in the position that corresponds to x_i in the control-part (whereas in case $\ell = \overline{x_i}$ is negated, this bit is not used). Furthermore, if literal ℓ occurs in clause c_j, then the weight has a 1-digit in the position corresponding to clause c_j in the verification-part. All other digits and bits are 0.
- For every literal $\ell \in \{y_i, \overline{y_i}\}$ there is a corresponding Y-item $Y(\ell)$. Its weight has a 1-digit in the position corresponding to variable y_i in the Y-part. If literal ℓ occurs in clause c_j, then the weight of $Y(\ell)$ has a 1-digit in the position corresponding to clause c_j in its verification-part. All other digits and bits are 0.
- For every clause c_j there are three C-items $C^k(c_j)$ with $k = 0, 1, 2$. The weight of item $C^k(c_j)$ has a 1-digit in the position corresponding to clause c_j in the clause-part, and a k-digit in the position corresponding to clause c_j in the verification-part. All other digits and bits are 0.
- Finally there are two dummy items D_0 and D_1 whose weights are $w(D_0) = U - 1$ and $w(D_1) = U + 2^s$. The integer U in these weights is defined as follows: it has a 3-digit in every position in the verification-part, a 1-digit in every position in the clause-part, Y-part, and X-part, and an all-zero control-part.

We will throughout refer to the $4s + 3t$ non-dummy items as XYC-items. To complete the description of the Captive-Queen instance, we define the weight bound $\Delta = 1$, the initial state $I_0 = \{D_0\}$ with $w(I_0) = U - 1$, and the goal state $I_1 = \{D_1\}$ with $w(I_1) = U + 2^s$.

▶ **Lemma 6.** *The constructed instance of* Captive-Queen *satisfies the following.*

(i) *If we add up the decimal representations of the weights of some subset J of XYC-items, then there will be no carry-overs from lower positions to higher positions in*

the verification-part, clause-part, Y-part, and X-part. Furthermore, there will be no carry-over from the control-part to the X-part.

(ii) *If $w(I_0) < V < w(I_1)$ holds for some integer V, then the verification-part, clause-part, Y-part, and X-part of V agree with the corresponding part of U. The control-part of V lies between 0 and $2^s - 1$.*

(iii) *If $w(I_0) < w(J) < w(I_1)$ holds for some item set J, then J contains no dummy items and hence solely consists of XYC-items.*

Proof. Statement (i) follows by looking into the digits and bits in our construction. Only X-items $X(x_i)$ for un-negated literals have non-zero control-part, and these control-parts altogether only add up to $2^s - 1$. Every position in the decimal representation of verification-part, clause-part, Y-part, or X-part is non-zero for at most five XYC-items. For statement (ii), note that $U \leq V \leq U + 2^s - 1$ and note that the control-part of U is 0.

For statement (iii), observe that all item weights in the instance are greater than 10^s. If set J contains dummy item D_0 then it must also contain some other item, and this brings $w(J)$ above $w(I_1)$. And if J contains dummy item D_1 then its weight is above $w(I_1)$. ◀

We now define a bijection between the 2^s integers V with $U \leq V \leq U + 2^s - 1$ on one side and the 2^s truth-settings of the Boolean variables in $X = \{x_1, \ldots, x_s\}$ on the other side. Since $0 \leq V - U \leq 2^s - 1$, the binary representation of $V - U$ consists of s bits. Then the ith bit (counted from the right end) specifies the truth-value of variable x_i in the corresponding truth-setting $T_V(X)$. Vice versa, any truth-setting for X can be interpreted as the binary representation of some integer where the value of x_i specifies the ith bit. By adding the value U to this integer, we get the number from the range $U, \ldots, U + 2^s - 1$ that corresponds to the truth-setting.

The following two lemmas will be proved in the full version of this paper.

▶ **Lemma 7.** *Let V be an integer with $U \leq V \leq U + 2^s - 1$. If there exists an item set J with $w(J) = V$, then there also exists a truth-setting $T(Y)$ for the Boolean variables in Y, such that formula $\phi(X, Y)$ is true under the combined truth-setting $T_V(X)$ and $T(Y)$.*

▶ **Lemma 8.** *Let V be an integer with $U \leq V \leq U + 2^s - 1$. If there exists a truth-setting $T(Y)$ for the Boolean variables in Y such that formula $\phi(X, Y)$ is true under the combined truth-setting $T_V(X)$ and $T(Y)$, then there exists an item set J with $w(J) = V$.*

Let us wrap things up. Assume that the constructed instance of Captive-Queen has answer YES. By Corollary 4 this is the case if and only if for all integers V in the range $w(I_0) < V < w(I_1)$ there exists an item set J with $w(J) = V$. By Lemma 7 and Lemma 8 this is the case if and only if for all truth-settings $T_V(X)$ with $U \leq V \leq U + 2^s - 1$ for the variables in X, there exists a truth setting for the variables in Y such that formula $\phi(X, Y)$ is true. And finally this exactly means that the considered instance of 2-Quantified 3-CNF-Sat has answer YES. Together with Lemma 5 this yields the main result of the paper.

▶ **Theorem 9.** *Problem* Captive-Queen *is Π_2^p-complete.* ◀

Our reduction establishes Π_2^p-hardness of Captive-Queen for the special case where $\Delta = 1$. If we multiply all item weights in our construction by a factor f, then we also derive Π_2^p-hardness for the cases where $\Delta = f$.

Finally let us settle the complexity of Captive-Queen-with-few-Moves

▶ **Theorem 10.** *Problem* Captive-Queen-with-few-Moves *is NP-complete.*

Proof. The NP-certificate consists of the at most m intermediate states that the system traverses while moving from the initial state to the goal state.

The NP-hardness proof is done by a reduction from the NP-hard Subset-Sum problem (see Garey & Johnson [3]): Given a sequence $q_1, \ldots, q_n$ of positive integers and a positive integer Q, is there an index-set $N \subseteq \{1, \ldots, n\}$ with $q(N) = Q$? Consider the item set $I = \{1, \ldots, n+2\}$ with $w(i) = 2q_i$ for $1 \leq i \leq n$, and with $w(n+1) = 2Q - 1$ and $w(n+2) = 2Q + 1$. The difference bound is $\Delta = 1$, the initial state is $I_0 = \{n+1\}$, the goal state is $I_1 = \{n+2\}$, and the bound on the number of moves is $m = 2$. The only way of moving from I_0 to I_1 is through a state $N \subseteq \{1, \ldots, n\}$ with $q(N) = Q$. ◀

4 Two highly structured special cases

In this section we analyze special cases of Captive-Queen where the weight sequence carries a strong combinatorial structure and therefore behaves nicely. We will show that for these special cases the (otherwise difficult) problems Captive-Queen and Captive-Queen-with-few-Moves become solvable in polynomial time. The following two auxiliary tools T1 and T2 (for an item set I with weights $w(i)$ for $i \in I$) form the main ingredients for our algorithms.

T1. Compute the maximum gap in the weight spectrum between two given bounds $w(I_0)$ and $w(I_1)$; this maximum gap is denoted by $\text{gap}(I, w, I_0, I_1)$.
T2. Compute the largest value W with $W \leq d$ in the weight spectrum; the corresponding value is denoted $W_{\max}(I, w, d)$.

▶ **Lemma 11.** *Consider a specially structured family of weight sequences for which the tools T1 and T2 can be implemented in polynomial time. Then for this family also the problems* Captive-Queen *and* Captive-Queen-with-few-Moves *can be solved in polynomial time.*

Proof. By Corollary 4.(ii) an instance of Captive-Queen has answer YES if and only if $\text{gap}(I, w, I_0, I_1) \leq \Delta$. Furthermore, an instance of Captive-Queen-with-few-Moves can be solved as follows. We let $d_0 = w(I_0)$ and then compute the auxiliary values $d_j = W_{\max}(I, w, d_{j-1} + \Delta)$ for $j = 1, \ldots, m$. The instance has answer YES if and only if $d_m \geq w(I_1)$. ◀

4.1 The case with super-increasing weight sequences

In this section we consider item sets $I = \{1, \ldots, n\}$ whose weights are super-increasing and hence satisfy the following inequalities. These conditions originate from the knapsack literature; see for instance Magazine, Nemhauser & Trotter [5].

$$w(1) + w(2) + \cdots + w(i-1) \;<\; w(i) \qquad \text{for } i = 1, \ldots, n. \tag{3}$$

With every subset $J \subseteq I$ we associate a binary number $\text{bin}(J) = b_n b_{n-1} \ldots b_2 b_1$ whose bits are defined as $b_i = 1$ if $i \in J$ and $b_i = 0$ if $i \notin J$. Furthermore we denote by J^+ the subset with $\text{bin}(J^+) = \text{bin}(J) + 1$ (in case $J \neq I$), and we denote by J^- the subset with $\text{bin}(J^-) = \text{bin}(J) - 1$ (in case $J \neq \emptyset$). It is easy to see (and also well-known) that $w(J) < w(K)$ holds if and only if $\text{bin}(J) < \text{bin}(K)$. Hence distinct subsets always have distinct weights, and the weight spectrum consists of 2^n pairwise distinct values. For any set $J \subseteq I$ with $\emptyset \neq J \neq I$, the three numbers $w(J^-)$, $w(J)$, $w(J^+)$ form three consecutive values in the weight spectrum.

Now let us discuss the gap between two consecutive values $w(J)$ and $w(J^+)$ (with $J \neq I$) in the weight spectrum. Let $k \in I$ be the smallest element that is not contained in J. Since $\text{bin}(J^+) = \text{bin}(J) + 1$, the set J^+ results from J by adding element k to it while simultaneously removing the elements $1, 2, \ldots, k-1$ from it. This yields that the gap length $w(J^+) - w(J)$ equals

$$G_k \;:=\; w(k) - \sum_{j=1}^{k-1} w(j). \tag{4}$$

Next consider an input I, w, I_0, I_1 for tool T1. Let α be the largest element in the symmetric difference of I_0 and I_1; then $\alpha \notin I_0$ and $\alpha \in I_1$. Define an intermediate set $I_{1/2}$ that agrees with I_0 and I_1 on all elements above α, that contains α, and that contains none of the elements below α. From now on we assume that $I_0 \neq I_{1/2}^-$ and $I_1 \neq I_{1/2}$, as the cases with equality are easily settled. The maximum gap between $w(I_0)$ and $w(I_1)$ either is the maximum gap between $w(I_0)$ and $w(I_{1/2}^-)$, or the $w(I_{1/2}^-)$ and $w(I_{1/2})$, or it is the maximum gap between $w(I_{1/2})$ and $w(I_1)$. Hence it is sufficient to determine these three gaps, and then to output the value of the largest one.

In order to analyze the first gap, let β be the largest element with $\beta \notin I_0$ that is strictly smaller than α. Since $I_{1/2}^-$ does contain β and also all the elements below β, the maximum gap between $w(I_0)$ and $w(I_{1/2}^-)$ equals $\max_{k \leq \beta} G_k$. The second gap between $w(I_{1/2}^-)$ and $w(I_{1/2})$ equals G_α. For the third gap, let γ be the largest element with $\gamma \in I_1$ that is strictly smaller than α. Since $I_{1/2}$ neither contains γ nor any of the elements below γ, the maximum gap between $w(I_{1/2})$ and $w(I_1)$ equals $\max_{k \leq \gamma} G_k$. This completes the polynomial time algorithm for tool T1.

A polynomial time algorithm for tool T2 can be found in the literature (Magazine, Nemhauser & Trotter [5]), and is based on a simple greedy approach. Consider a knapsack of size d, and repeatedly pack the largest unpacked item weight into this knapsack. When no further item fits into the knapsack, the overall weight in the knapsack equals $W_{\max}(I, w, d)$.

▶ **Theorem 12.** *Problems* Captive-Queen *and* Captive-Queen-with-few-Moves *can be solved in polynomial time, if the weight sequence is super-increasing.* ◀

4.2 The case with divisible weight sequences

In this section we consider item sets $I = \{1, \ldots, n\}$ whose weights satisfy the following divisibility conditions. These conditions come from the knapsack and packing literature where they have been investigated thoroughly; see for instance Pochet & Wolsey [8] and Coffman, Garey & Johnson [1].

$$w(i) \mid w(i+1) \qquad \text{for } i = 1, \ldots, n-1. \tag{5}$$

Our first goal is to design a polynomial time algorithm for tool T1. We distinguish two cases. First assume that $w(1) > 1$. Then (5) implies that all item weights are divisible by $w(1)$. In this case we define new item weights $w'(i) = w(i)/w(1)$, and observe that

$$\text{gap}(I, w, I_0, I_1) \;=\; w(1) \cdot \text{gap}(I, w', I_0, I_1). \tag{6}$$

Next assume that $w(1) = 1$ holds, and define ℓ as the largest integer with $w(\ell) = 1$. Let $I' = \{\ell + 1, \ldots, n\}$ contain the items of weight greater than 1, let w' be the restriction of the weights w from I to I', and let $I_0' = I_0 \cap I'$ and $I_1' = I_1 \cap I'$. Then

$$\text{gap}(I, w, I_0, I_1) \;=\; \max\left\{\text{gap}(I', w', I_0', I_1') - \ell,\; 1\right\}. \tag{7}$$

Note that $I' = \emptyset$ in (7) yields $\text{gap}(I, w, I_0, I_1) = 1$. The two formulas in (6) and (7) yield a recursive procedure for computing $\text{gap}(I, w, I_0, I_1)$. The running time of the procedure is polynomial, and with a little effort can even be made linear in n.

Tool T2 is available in the literature (Coffman, Garey & Johnson [1]), and follows the same greedy approach as tool T2 for super-increasing weight sequences.

▶ **Theorem 13.** *Problems* Captive-Queen *and* Captive-Queen-with-few-Moves *can be solved in polynomial time, if the weight sequence is divisible.* ◀

5 Analysis of four related problems

We will now discuss some further properties of the discrete system that underlies the Captive-Queen problem. Recall that every state of the system corresponds to a subset of items, and that the system can move from state J directly to state K if (1) respectively (2) is satisfied. A state J is *isolated*, if there are no other states reachable from J. A system is *fully connected*, if every state is reachable from every other state. Equivalently, a system is fully connected if and only if the state $\emptyset$ is reachable from the state I.

Figure 3 lists four algorithmic problems that are formulated around isolated states and fully connected systems. The full version of this paper will show that

- Problem Isolated-State is coNP-complete;
- Problem All-States-Isolated is coNP-complete;
- Problem Some-State-Isolated can be decided in polynomial time;
- Problem Fully-Connected can be solved in polynomial time.

Furthermore, the full version will show that all these problems are easy, if the weight sequence is super-increasing (see Section 4.1) or divisible (see Section 4.2).

Problem: Isolated-State
Instance: An item set I; weights $w(i)$ for $i \in I$; a bound Δ; a subset $J \subseteq I$.
Question: Is the state J isolated?

Problem: All-States-Isolated
Instance: An item set I; weights $w(i)$ for $i \in I$; a bound Δ.
Question: Are all states in this system isolated?

Problem: Some-State-Isolated
Instance: An item set I; weights $w(i)$ for $i \in I$; a bound Δ.
Question: Does this system contain some isolated state?

Problem: Fully-Connected
Instance: An item set I; weights $w(i)$ for $i \in I$; a bound Δ.
Question: Is this system fully connected?

Figure 3 The algorithmic problems discussed in Section 5.

References

1 E.G. Coffman Jr., M.R. Garey, and D.S. Johnson (1987). Bin packing with divisible item sizes. *Journal of Complexity 3*, 406–428.

2 T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein (2001). *Introduction to Algorithms*. MIT Press.

3 M.R. Garey and D.S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco.

4 H. Kellerer, U. Pferschy, and D. Pisinger (2004). *Knapsack problems*. Springer Verlag, Berlin.

5 M. Magazine, G.L. Nemhauser, and L.E. Trotter (1975). When the greedy solution solves a class of knapsack problems. *Operations Research 23*, 207–217.

6 S. Martello and P. Toth (1990). *Knapsack problems: Algorithms and computer implementations*. John Wiley & Sons, Chichester.

7 C.H. Papadimitriou(1994). *Computational Complexity*. Addison-Wesley.

8 Y. Pochet and L.A. Wolsey (1995). Integer knapsack and flow covers with divisible coefficients: Polyhedra, optimization and separation. *Discrete Applied Mathematics 59*, 57–74.

9 L.J. Stockmeyer (1977). The polynomial-time hierarchy. *Theoretical Computer Science 3*, 1–22.

10 E. Wakeling (1995). *Rediscovered Lewis Carroll puzzles*. Courier Dover Publications.

11 G.J. Woeginger and Z. Yu (1992). On the equal-subset-sum problem. *Information Processing Letters 42*, 299–302.

On Computing Pareto Stable Assignments

Ning Chen[1]

1 **Nanyang Technological University**
Singapore
ningc@ntu.edu.sg

Abstract

Assignment between two parties in a two-sided matching market has been one of the central questions studied in economics, due to its extensive applications, focusing on different solution concepts with different objectives. One of the most important and well-studied ones is that of stability, proposed by Gale and Shapley [8], which captures fairness condition in a model where every individual in the market has a preference of the other side. When the preferences have indifferences (i.e., ties), a stable outcome need not be Pareto efficient, causing a loss in efficiency. The solution concept Pareto stability, which requires both stability and Pareto efficiency, offers a refinement of the solution concept stability in the sense that it captures both fairness and efficiency.

We study the algorithmic question of computing a Pareto stable assignment in a many-to-many matching market model, where both sides of the market can have multiunit capacities (i.e., demands) and can be matched with multiple partners given the capacity constraints. We provide an algorithm to efficiently construct an assignment that is simultaneously stable and Pareto efficient; our result immediately implies the existence of a Pareto stable assignment for this model.

1998 ACM Subject Classification F.2.0 [Analysis of Algorithms and Problem Complexity] General

Keywords and phrases Algorithm, stable matching, Pareto efficiency

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.384

1 Introduction

Two-sided matching markets have been extensively studied since the seminal work of Gale and Shapley on stable marriage [8], where there are a set of men and women, each with a strict preference ranking over members of the other side. A matching between the men and women is *stable* if there is no man-woman pair who both strictly prefer each other to their current partners. The concept of stability captures fairness condition for market participants and has had enormous influence on the design of real world matching markets [18]. The original marriage model, as well as many of its generalizations, have been thoroughly investigated.

A practical reality of matching markets is ties, or indifferences: Agents may not be able to strictly rank their prospective partners, i.e., they might be indifferent among some of them. The introduction of ties into preference lists dramatically changes the properties and structure of the set of stable matchings. For instance, man or woman-optimal stable matchings are no longer well-defined [19], and stable matchings need not all have the same cardinality. The problem of finding a maximum cardinality stable matching becomes NP-hard [11, 16], and much work has focused on finding approximation solutions [13, 17]. In addition, arguably more importantly, *stability no longer guarantees Pareto efficiency* (roughly speaking, Pareto efficiency means that no other feasible solution exists that improves some agent without

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 384–395

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

hurting everyone else)[1], an observation that has received much attention in the economics literature (e.g., [2, 22, 6, 1, 7]). In particular, as Erdil and Ergin [6] demonstrate, simply using a matching returned by the Gale-Shapley deferred acceptance algorithm can cause quite a severe loss in efficiency.

To capture both fairness and efficiency, Sotomayor [22] suggests *Pareto stability* (i.e., both stable and Pareto efficient) as a natural solution concept for matching markets in the presence of indifferences. A natural question is then whether such a matching always exists, and how to find one efficiently. Note that the presence of ties in the preference lists cannot be addressed by the standard trick of introducing small perturbations: If ties are broken arbitrarily, the set of stable matchings with respect to the new strict preferences can be strictly smaller than the set of stable matchings with respect to the original preferences with ties — that is, artificial tiebreaking does not preserve the set of stable matchings in the original problem, thus, may not generate a Pareto efficient matching.

This question has recently been addressed by Erdil and Ergin [6, 7] for the *many-to-one* matching model, where one side of the market can have multi-unit capacity. The authors showed an algorithm to find a Pareto stable assignment, when an agent's preference over subsets of neighbors is the natural partial order derived from preferences over individual neighbors. In recent years, there are a growing number of instances of *many-to-many* matching markets, such as online labor markets, course assignments, and the UK medical intern markets, where agents on both sides might want to transact with multiple agents from the other side. In course assignments, for instance, students may register for multiple courses and have preferences over them; on the other hand, courses may have implicit preferences over students according to their, e.g., years of study, majors.

The generalization to multi-unit demand on both sides is nontrivial: The many-to-many stable matching problem behaves rather differently from the many-to-one and one-to-one models in terms of the properties and structure of the set of solutions [19, 5, 21, 22]. In this paper, we study Pareto stability with indifference for many-to-many matching, by considering the natural generalization of the Erdil and Ergin model to many-to-many market: What happens when agents have the same preference model over subsets of acceptable partners as in [6, 7], but agents on both sides can have capacities greater than one? Our main result is the following:

Theorem. *For many-to-many matching markets with indifferences, a Pareto stable assignment always exists and can be computed in polynomial time.*

1.1 Algorithmic Ideas

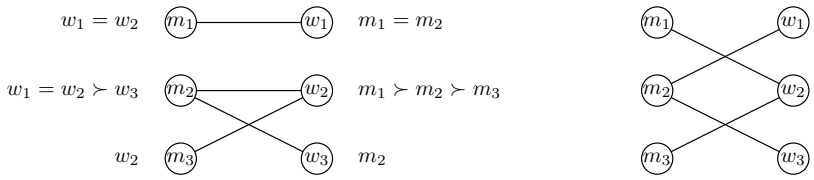
The algorithm of Erdil and Ergin [6, 7] for the many-to-one matching model depends on two observations: First, an assignment has a Pareto improvement (i.e., another assignment where no one gets worse off and at least one agent gets better off) only if the assignment graph does have an augmenting path or cycle (formal definitions refer to Section 2). Second, more critically, any Pareto improvement to a stable assignment preserves stability. These observations immediately imply an algorithm to find a Pareto stable assignment: Starting from any stable assignment, keep making Pareto improvements by eliminating augmenting

[1] For example, there are two men m_1, m_2 and two women w_1, w_2, where m_1 strictly prefers w_1 to w_2, but all others are indifferent amongst their possible partners. The matching $(m_1, w_2), (m_2, w_1)$ is stable, but not Pareto efficient since m_1 can be reassigned to w_1 and m_2 to w_2 without making anyone worse off.

paths and cycles until none remain, and the resulting matching will be both stable and Pareto efficient.

In the many-to-many setting, however, while the first property still holds, we observe that the second critical property fails: A Pareto improvement to a stable assignment need not preserve stability, as the following example shows.

▶ **Example 1** (Pareto improvement does not preserve stability). *Consider the example in the following figure, where m_2 and w_2 have capacity two each and other agents all have unit capacity. Preferences are specified next to each node, e.g., m_2 is indifferent between w_1 and w_2 and prefer both of them to w_3.*



It can be seen that the assignment on the left is stable, and the assignment on the right is a Pareto improvement where w_2 strictly improves her assignment and no one gets worse off. However, the assignment on the right is unstable as m_2 and w_2 would like to match with each other rather than w_3 and m_3 respectively, i.e., it is a blocking pair.

Since Pareto improvements need not preserve stability, all previous approaches (e.g., [6, 7, 4]) computing a Pareto stable assignment in variant models fail. Further, even the existence of a Pareto stable assignment in the many-to-many setting is unclear. We will give an explicit algorithm to compute a Pareto stable assignment, which implies the existence immediately. The above example already shows that the approach of starting with an arbitrary stable assignment and making Pareto improvements will not work, since this need not preserve stability. Further, for the given stable assignment in the above example (left), there is only one Pareto improvement (right); thus, the problem cannot be solved by a careful selection of Pareto improvements.

Instead of using a stability preserving Pareto improvement approach, our algorithm builds on the idea of Roth and Vande Vate [20], who provide an alternative to the deferred acceptance algorithm to compute a stable (one-to-one) matching. Their algorithm can be interpreted as follows: Assume that all women are present at the beginning, and men 'arrive' one by one. We start with the empty matching. When a new man m arrives, match him to a most preferred woman w with whom he forms a blocking pair, if any; if this woman was already matched to a man m', set m' free and consider him as the next arriving man; the algorithm runs iteratively until all men have arrived. Since every woman who changes her partner in this process gets a strict improvement and no woman ever becomes worse off, the algorithm terminates; the final matching is stable, since by construction the matching at every man's arrival is stable.

Our algorithm, like [20], assumes all women are available and considers men one by one (precisely, increases their capacities unit by unit). When the capacity of a man is increased by one, we do a sequence of reassignments to guarantee stability (with respect to the current considered capacities). Further, we ensure that no woman ever becomes worse off, and that some woman strictly improves her assignment in each phase. The algorithm hence will eventually terminate and lead to a stable matching; it remains to consider Pareto efficiency.

An important idea in our algorithm to derive Pareto efficiency is that in the process of reassignments, no augmenting cycles have ever been introduced in the matching; but, on the other hand, we allow the existence of augmenting paths. The key component of our algorithm is a subroutine for eliminating augmenting paths while preserving stability (and introducing no augmenting cycles). Having constructed a matching which is stable and contains no augmenting cycles, we apply the subroutine to eliminate augmenting paths in a stability preserving fashion, which finally yields a Pareto stable matching.

1.2 Related Work

There is a vast literature studying various aspects of the original stable marriage model of Gale and Shapley [8], as well as many of its variants. For a nice review of the very large economics literature on the subject, see the book by Roth and Sotomayor [19] and the survey by Roth [18]; for an introduction to algorithmic and computational issues, see, for instance, the textbook by Gusfield and Irving [9] and the survey by Iwama and Miyazaki [12].

When preferences have indifferences, Irving [14] defined two different notions, weak stability and strong stability, to capture different levels of stability. These two solution concepts, while conceptually similar, have rather different properties. The stability considered in our paper corresponds to weak stability. The computer science literature has largely focused on, e.g., approximating the maximum cardinality weakly stable matching (e.g., [13, 17]) or computing strongly stable matchings (e.g. [14, 15, 10, 3]).

Another related work is [4] which also considers Pareto stable solutions in many-to-many settings; however, in the work of [4], every pair of agents can transact any number of units (e.g., money transfer) which is very different from the present paper where at most one unit can be assigned. Specifically, in the model of [4], the stability-preserving and augmenting path/cycle elimination properties still hold; thus, a Pareto stable solution exists trivially and the algorithm in Erdil and Ergin [6, 7] can be applied directly. The solution structures, algorithm ideas and technical details in our paper are all quite different from [6, 7, 4].

2 Preliminaries

In a two-sided marketplace, let M be the set of men and W be the set of women. Throughout this paper, we will use $m \in M$ to denote a man and $w \in W$ to denote a woman, and use $x, y, z \in M \cup W$ to denote any individual agent (man or woman). For each agent $x \in M \cup W$, let $c_x \in \mathbb{N}$ be his/her capacity, which is the maximum number of agents on the opposite side that can be matched to x. The presence of capacities allows us to assume, without loss of generality, that $|M| = |W| = n$, as dummy agents with $c_x = 0$ can be added to the market.

Each man $m \in M$ has a *preference* list P_m ranking individual women, denoted by $\succ$ and $=$, where $w_1 \succ w_2$ means that the man (strictly) prefers w_1 to w_2, and $w_1 = w_2$ means that m is indifferent between them. We say m weakly prefers w_1 to w_2 if either $w_1 \succ w_2$ or $w_1 = w_2$, denoted by $w_1 \succeq w_2$. Every two women in P_m are comparable and the preference is assumed to be transitive. The preference P_m gives individual women that are acceptable to m, and it may only contain a partial list of women (i.e., m does not want to be matched with any woman that is not on the list). For example, a possible preference list for m is $P_m : (w_1 = w_2 \succ w_3 = w_5)$: here, m is indifferent between w_1 and w_2, and prefers either of them to w_3, w_5, amongst which m is indifferent; he finds all other partners unacceptable. The preference list P_w for each woman $w \in W$ is defined similarly. Let $E = \{(m, w) \mid m \in P_w, w \in P_m\}$ be the set of mutually acceptable pairs. The problem then

can be encoded as a bipartite graph $(M, W; E)$ where every vertex has a capacity and a preference over its neighbors.

Notice that the preference lists P_m and P_w defined above are over individual neighbors. Since agents can have capacities greater than one, we also need to define preferences over subsets of neighbors. For this, we adopt the preference model used by Erdil and Ergin [6, 7] in their work for many-to-one markets — the preference ordering over individuals defines a natural ranking over subsets of acceptable partners — given a subset $S \subseteq W$ and two women $w, w' \notin S$, m prefers $S \cup \{w\}$ to $S \cup \{w'\}$ if and only if m prefers w to w' (it is allowed that $w, w' = \emptyset$). In addition, the preference is transitive, i.e., if m prefers S_1 to S_2 and S_2 to S_3, it prefers S_1 to S_3 as well. The preferences of women are defined similarly. For example, if $P_m = (w_1 \succ w_2 \succ w_3 \succ w_4)$, then m prefers $\{w_1, w_2, w_3\}$ to $\{w_1, w_2, w_4\}$, also m prefers $\{w_1, w_3\}$ to $\{w_2\}$ (via $\{w_2, w_3\}$ in the middle). Note that this preference over subsets only constitutes a partial order — specifically, some subsets may not be comparable — for example, m cannot compare (or equivalently, is indifferent between) the sets $\{w_1, w_4\}$ and $\{w_2, w_3\}$.[2]

Given the preferences of all agents, our objective is to establish a multi-unit pairing between men and women, called an *assignment* (or b-matching). An assignment is denoted by $\mu = (\mu_{mw})_{m \in M, w \in W}$, where $\mu_{mw} = 1$ means that m and w are matched and $\mu_{mw} = 0$ otherwise. A *feasible* assignment is one that satisfies the following conditions: $\sum_w \mu_{mw} \leq c_m$ and $\sum_m \mu_{mw} \leq c_w$, and $\mu_{mw} = 1$ only if $(m, w) \in E$ (i.e., m and w are mutually acceptable). All assignments considered in this paper are feasible.

2.1 Solution Concepts

We will consider the following solution concepts.

▶ **Definition 2** (Stability)**.** We say that a feasible assignment $\mu = (\mu_{mw})$ is (pairwise) stable if there is no pair $(m, w) \in E$ (called a blocking pair), $\mu_{mw} = 0$, satisfying one of the following conditions:

- Both m and w have leftover capacity;
- m has leftover capacity and there is m', $\mu_{m'w} = 1$, such that w strictly prefers m to m'; or w has capacity remaining and there is w', $\mu_{mw'} = 1$, such that m prefers w to w';
- There are m' and w', $\mu_{mw'} = 1$ and $\mu_{m'w} = 1$, such that m strictly prefers w to w' and w strictly prefers m to m'.

Note that both members of a blocking pair are able to improve their assignments respectively by matching with each other (and possibly breaking some of the current assignments). A stable assignment always exists, and can be found using a variant of Gale-Shapley's deferred acceptance algorithm [8] for computing stable matchings (by making c_x copies for each individual $x \in M \cup W$ with the same preference list).

We next define Pareto efficiency. Roughly speaking, an assignment is Pareto efficient if there is no other feasible assignment where no agent is worse off, and at least one agent is strictly better off. The formal definition is given below.

[2] The preference we consider is called *responsive* preference in economics (see, e.g., [19]). This model of preferences with multi-unit capacity is both simple, since agents continue to only express preferences over individuals, and is arguably natural for settings where the benefit from a partner to an agent does not depend upon the agent's remaining partners.

▶ **Definition 3** (Pareto efficiency)**.** Given a feasible assignment $\mu = (\mu_{mw})$, let $S_x(\mu)$ be the subset of individuals assigned to x in μ. We say that $\mu' = (\mu'_{mw})$ is a Pareto improvement of μ if for all $x \in M \cup W$, x weakly prefers $S_x(\mu')$ to $S_x(\mu)$, and the preference is strict for at least one agent. An assignment μ is called Pareto efficient if it does not have any Pareto improvement.

Recall from Introduction that when preference lists contain ties, a stable assignment need not be Pareto efficient. This leads naturally to the concept of Pareto stability [22], which combines both Pareto efficiency and stability to provide a stronger solution concept to choose from amongst the set of feasible assignments.

▶ **Definition 4** (Pareto stability)**.** A feasible assignment is *Pareto stable* if it is both stable and Pareto efficient.

2.2 Characterization of Pareto Efficiency

Given the connection between matching and network flow, it is not surprising that the existence of augmenting paths and cycles in an assignment is closely related to whether it can be improved, i.e., its Pareto efficiency. The main difference in the context of stable assignment is that nodes have preferences in addition to capacities; thus, augmenting paths and cycles must improve not just the size of an assignment, but also its quality, as determined by node preferences. We first define augmenting path and cycle in the context of stable assignment.

▶ **Definition 5** (Augmenting Path)**.** Given an assignment $\mu = (\mu_{mw})$, we say that

$$[m_0, w_1, m_1, \ldots, w_\ell, m_\ell, w_{\ell+1}]$$

is an augmenting path if (i) $\sum_w \mu_{m_0 w} < c_{m_0}$ and $\sum_m \mu_{m w_{\ell+1}} < c_{w_{\ell+1}}$, (ii) $\mu_{m_k w_k} = 1$ and $\mu_{m_{k-1} w_k} = 0$ for all k, and (iii) m_k weakly prefers w_{k+1} to w_k and w_k weakly prefers m_{k-1} to m_k.

The first condition says that the capacities of m_0 and $w_{\ell+1}$ are not exhausted. The second condition says that pairs alternatively are not and are in the current assignment μ along the path. The last condition ensures that we are able to get a Pareto improvement by reassigning matches according to the augmenting path. That is, removing all pairs (m_k, w_k) and matching all pairs (m_k, w_{k+1}) give a feasible assignment, which is a Pareto improvement over μ (where no one is worse off and m_0 and $w_{\ell+1}$ are better off).

▶ **Definition 6** (Augmenting Cycle)**.** Given an assignment $\mu = (\mu_{ij})$, we say that

$$[m_1, w_2, m_2, \ldots, w_\ell, m_\ell, w_1, m_1]$$

is an augmenting cycle if (i) $\mu_{m_k w_k} = 1$ and $\mu_{m_k w_{k+1}} = 0$ for all k (where $w_{\ell+1} = w_1$) (ii) m_k weakly prefers w_{k+1} to w_k and w_k weakly prefers m_{k-1} to m_k, and at least one of these preferences is strict.

Again, we are able to match all pairs (m_k, w_{k+1}) and unmatch all pairs (m_k, w_k) in an augmenting cycle to get a Pareto improvement. For a given assignment, an augmenting path or cycle can be found easily by a network flow approach.

The following lemma characterizes the relation between stable assignment and augmenting path and cycle (its proof is the same as the one for many-to-one matching market [6]).

▶ **Lemma 7.** *A feasible assignment is Pareto efficient if and only if it has no augmenting path or cycle.*

3 Algorithm

In this section, we will give an efficient algorithm to compute a Pareto stable assignment. By the above characterization Lemma 7, it suffices to find a stable assignment without containing any augmenting path and cycle. We will apply the idea of the Roth and Vande Vate algorithm [20] that computes a stable one-to-one matching as the high level structure of our algorithm: Initially all individuals are available; women are with *full* capacities and men are with *null* capacity. We increase capacities of men unit by unit, and do a number of reassignments in the process. In the course of the algorithm, the current assignment always has the following invariants (with respect to the current capacities):

- **Stability preserving:** *it is always stable.*
- **No augmenting cycle:** *it does not contain any augmenting cycle.*
- **Women improving:** *the assignments of all women do not get worse off and overall keep improving (this implies that the algorithm always terminates).*

Why do we need to maintain the invariant that the algorithm contains no augmenting cycles, whereas it is allowed to have augmenting paths? Observe that the reason that a Pareto improvement may not preserve stability is that the path or cycle corresponding to the Pareto improvement contains a matched pair (m, w) where both m and w are also matched to a less preferred agent, say w' and m'. When the match (m, w) is removed in the reassignment process of the augmenting path/cycle, even though m and w could receive better partners in the path or cycle, they will prefer to be matched to each other instead of w' and m' respectively. For augmenting path, however, we can always start reassignment from one side of the path (say, the man), and stop proceeding along the path when we reach such a woman w (then (m', w) is unmatched and the process restarts). In this stability-preserving process, a woman becomes strictly better off. However, for the pair (m, w) in an augmenting cycle, we would need to release both (m', w) and (m, w') to preserve stability. That is, we would no longer have the monotonically improving property for women's assignments, which is critical to the analysis of the algorithm.

The high-level structure of the algorithm is described below:

PARETO-STABLE-ALG

1. `Initialization`
   - `there are no assigned edges (i.e.,` $\mu = 0$`) between` M `and` W
   - `all women have their full capacities available`
   - `let` $\mathbf{d} = (d_m)_{m \in M}$ `be a virtual capacity vector of men; initially` $d_m = 0$ `for` $m \in M$
2. `While there is` $m \in M$ `such that` $d_m < c_m$
   - `run` INCREASE-CAP($\mathbf{d}$)
3. `While there is an augmenting path` P
   - `run` ELIMINATE-PATH(P)
4. `Return the final assignment`

Note that in the algorithm, $\mu = (\mu_{mw})_{m \in M, w \in W}$ and $(d_m)_{m \in M}$ are global variables in both subroutines. The first subroutine, INCREASE-CAP, increases the virtual capacity of a man by one and does a number of reassignments to ensure the three invariants listed above (in particular, it guarantees that the assignment is stable for the increased virtual capacity vector). The second subroutine, ELIMINATE-PATH, eliminates all possible augmenting paths

to derive a Pareto efficient assignment in a stability preserving fashion. The two subroutines are not completely independent: We may call the second subroutine in the process of the first one, and vice versa. After all augmenting paths have been eliminated, by Lemma 7, the returned assignment is Pareto stable.

While the algorithm may look a bit complicated as the two subroutines may call each other, the fact that no women ever get worse off in the process implies a simple, but critical, structure of the algorithm: we iteratively do a sequence of reassignments to improve women's assignments while preserving stability and containing no augmenting cycle. If at any moment in the algorithm a woman's assignment gets strictly improved, no matter at which stage the algorithm is, we terminate that thread immediately and go to Step (2) of the main algorithm to repeat the process given the current virtual capacity vector $\mathbf{d}$. Such monotonically improving property is crucial to the analysis of the algorithm.

We will describe the two subroutines in detail in the following subsections. (All discussions are with respect to the considered virtual capacity vector.) In the algorithm, for any (augmenting) cycle C and a pair $(m, w) \in C$, we use $C \setminus \{(m, w)\}$ to denote the path by removing pair (m, w) from C.

3.1 Subroutine One: Capacity Increment

The first subroutine that increases virtual capacities of the men is the following.

INCREASE-CAP($\mathbf{d}$)

1. Pick an arbitrary man m with $d_m < c_m$
2. Let $d_m \leftarrow d_m + 1$, i.e., increase the virtual capacity of m by one
3. Let $S = \{w \mid (m, w)$ is a blocking pair$\}$
4. Let $T = \{w \in S \mid m$ prefers $w \succeq w'$ for any $w' \in S\}$
5. If $T = \emptyset$ (i.e., there is no blocking pair), return
6. Otherwise
   a. If there exists $w \in T$ such that adding match (m, w) does not introduce any augmenting cycle
      - pick such a woman w'
      - add match (m, w')
   b. Otherwise
      - pick an arbitrary $w' \in T$
      - let C be a potential augmenting cycle by adding (m, w')
      - let $P = \left[m \xrightarrow{C \setminus \{(m,w')\}} w' \right]$ be the path from m to w' through $C \setminus \{(m, w')\}$
      - run ELIMINATE-PATH(P)
   c. If w' (defined either in Step (6.a) or (6.b)) is over-matched (i.e., matched to more than $c_{w'}$ neighbors)
      - let m' be a least preferred man matched to w' where deleting (m', w') does not introduce an augmenting cycle
      - delete match (m', w')
      - let $d_{m'} \leftarrow d_{m'} - 1$
      - return
   d. Otherwise, return

When the virtual capacity of m is increased by one, there might be some blocking pairs, among which the subroutine tries to match m to one that he prefers most ($w' \in T$ in the above description). However, this could introduce potential augmenting cycles (Step 6(b)).

Instead of matching m and w' directly, the subroutine considers a potential augmenting cycle C incurred by (m, w') and tries to do reassignments according the other path from m to w' along the cycle. Finally, if w' is over-matched, then we delete one of her least preferred assignments without incurring any augmenting cycles and delete the virtual capacity of that man by one. This guarantees that the assignment remains stable, and the assignment of w' strictly improves.

The existence of m' in Step 6(c) is guaranteed by the following lemma.

▶ **Lemma 8.** *Given a stable assignment without augmenting cycles, for any woman w let $S \subseteq M$ be the subset of men matched to w to whom w is least preferred. Then there is $m \in S$ such that deleting match (m, w) does not introduce any augmenting cycle.*

3.2 Subroutine Two: Augmenting Path Elimination

Consider any given stable assignment, assume that there is an augmenting path $P = [m_0, w_1, m_1, \ldots, w_\ell, m_\ell, w_{\ell+1}]$, where (m_i, w_i) is in the assignment and (m_i, w_{i+1}) is not. Note that it is possible that an individual x (either a man or a woman) or a pair (x, y) appears more than once in P. In this subsection, when we refer to an individual $x \in P$ or a pair $(x, y) \in P$, we denote the corresponding one at that position of P.

Before describing the subroutine, we will first consider a truncation process, which deletes some pairs in a given augmenting path according to different appearances of the same agent and will will be used in the subroutine.

3.2.1 Truncation.

For a given augmenting path P, we consider the following truncation function.

TRUNC-PATH(P)

1. **while** one of the following "if" conditions holds
   - If there is m such that $P = [\ldots, m, w_1, \ldots, w_2, m, \ldots]$ and m weakly prefers w_1 to w_2
     - truncate $P = [\ldots, m, \cancel{(w_1, \ldots, w_2, m,}\ \ldots]$
   - If there is w such that $P = [\ldots, w, m_1, \ldots, m_2, w, \ldots]$ and w weakly prefers m_2 to m_1
     - truncate $P = [\ldots, w, \cancel{(m_1, \ldots, m_2, w,}\ \ldots]$
2. Return path P

It can be seen that if TRUNC-PATH(P) is executed, by the rules of the truncation, no pair (x, y) can appear more than once after truncation. However, it is still possible that an individual appears more than once (e.g., when m strictly prefers w_2 to w_1, we do not truncate the two occurrences of m). The truncation process is necessary in our algorithm; in particular, it is important to the analysis of termination of the algorithm.

We have the following observation.

▶ **Lemma 9.** *For any given augmenting path P,* TRUNC-PATH(P) *returns an augmenting path as well.*

3.2.2 Elimination.

We next describe the subroutine to eliminate augmenting paths while preserving the three invariants listed at the beginning of the section. Note that for any augmenting path, its one

side must be a man and the other side must be a woman. The subroutine starts from the man side and considers pairs one by one. Hence, for any man-woman pair in the path, the objective is to match them; and for any woman-man pair in the path, the objective is to unmatch them.

ELIMINATE-PATH(P)

1. Assume $P = [m^*, w_1, m_1, \ldots, w^*]$
2. Let $e = (m^*, w_1)$ be the first pair on path P
3. **while** $e \neq \emptyset$
   - If e is not a match (i.e., $e = (m, w)$)
     - if adding match (m, w) does not introduce an augmenting cycle
       - a. add match (m, w)
       - b. if w is not over-matched, return
       - c. if w strictly prefers m to a current partners
         - let m' be a least preferred man matched to w where deleting (m', w) does not introduce an augmenting cycle (by Lemma 8, such m' exists)
         - delete match (m', w)
         - let $d_{m'} \leftarrow d_{m'} - 1$
         - return to Step 2 of the main algorithm PARETO-STABLE-ALG to run INCREASE-CAP
       - d. else let e be the next pair after (m, w) in P
     - otherwise
       - e. let $C = [m, w'_1, m'_1, \ldots, w'_k, m'_k, w, m]$ be such a potential cycle if adding (m, w)
       - f. expand $P = \left[m^*, \ldots, m, w'_1 \xrightarrow{C\setminus\{(m,w)\}} m'_k, w, \ldots, w^*\right]$
       - g. truncate $P = \left[m^*, \ldots, \text{TRUNC-PATH}\left(m, w'_1 \xrightarrow{C\setminus\{(m,w)\}} m'_k, w, \ldots, w^*\right)\right]$
       - h. let e be the first pair returned by the TRUNC-PATH
   - If e is a match (i.e., $e = (w, m)$)
     - if deleting match (w, m) does not introduce an augmenting cycle
       - i. delete match (w, m)
       - j. let e be the next pair after (w, m) in P
     - otherwise
       - k. run the above Steps (e,f,g,h) (switching the notations of m and w (except m^* and w^*))

The subroutine tries to add and delete matches one by one along pairs in the path P. If the current considered pair is a man-woman pair (i.e., $e = (m, w)$), the subroutines matches them if it does not introduce any augmenting cycle. If the assignment of w is strictly improved (i.e., the condition in Step (3.b) or (3.c) is satisfied), the subroutine terminates. Note that at this point the subroutine may not completely eliminate the augmenting path, however, the overall assignment of the woman gets strictly improved and the process restarts at the capacity increment stage. If matching m and w will introduce a potential augmenting cycle, instead of adding the match directly, the subroutine takes a "detour" and considers the other path from m to w along the cycle and expands it to the path P (Step 3(f); by the following Lemma 10, it is a valid expansion). Then the subroutine will do a truncation from m to the end of the path P and restarts the process by considering the first pair returned by the truncation (its first individual must be m). The subroutine performs similarly if the considered pair is a woman-man pair.

We first establish the following observations.

▶ **Lemma 10.** *The expansion of path P in Step (3.f) is a well-defined augmenting path.*

We have the following key claim, which implies that the subroutine always terminates.

▶ **Lemma 11** (Main). *The subroutine* Eliminate-Path(P) *terminates in finite number of steps for any augmenting path P.*

3.3 Analysis of the Algorithm

Again, the high level structure of the algorithm is to increase capacities of men and eliminate augmenting paths. While the algorithm may look involved, as the virtual capacity is not always monotonically increasing (e.g., in Step 6(c) of Increase-Cap and Step 3(c) of Eliminate-Path, we actually need to reduce the virtual capacities) and two subroutines may call each other, there is a simple, but crucial, idea behind the algorithm: the assignments of women keep improving (this is the exact reason that we do not want to introduce any augmenting cycle in the course of the algorithm). Therefore, at any moment of the algorithm, if a woman's assignment gets improved (e.g., Step 6(c) of Increase-Cap and Step 3(b), 3(c) of Eliminate-Path), the algorithm will abandon the current subroutine and restart the whole process (i.e., capacity increment and augmenting path elimination) starting from the current virtual capacity vector. Since every woman can improve her assignment at most n^2 times (as her capacity is at most n and every unit capacity can be improved at most n times), the whole algorithm will terminate.

It is easy to see that the three invariants listed at the beginning of the section are maintained in the course of the algorithm. Indeed, the last two (no augmenting cycle and women not worse off) hold trivially as they are guaranteed by the algorithm itself. For stability, in the subroutine Increase-Cap, when increasing the virtual capacity of m by one, we try to match m with a most preferred woman w where (m, w) forms a blocking pair. If w is not over-matched, then the resulting assignment is still stable. Otherwise, we delete a match (m', w) where m' is a least preferred man matched to w and reduce the virtual capacity of m' by one (Step (6.c) of Increase-Cap); this implies that the resulting assignment is still stable with respect to the new capacity vector. For the second subroutine Eliminate-Path, stability comes from the definition of augmenting path and the fact that when we delete a match (w, m), we know that m must be a least preferred man matched to w and w was over-matched (otherwise, when we add the match right before (w, m), the assignment of w gets strictly improved and the subroutine will run Step (3.b) or (3.c) to terminate). Therefore, the final returned assignment is stable.

By the rule of the algorithm Pareto-Stable-Alg, when it terminates there is no augmenting path. By the invariant that there is no augmenting cycle, we know that the returned assignment is Pareto efficient. We conclude with the following result.

▶ **Theorem 12.** *The algorithm* Pareto-Stable-Alg *computes a Pareto stable assignment in polynomial time.*

Acknowledgements

I am grateful for Arpita Ghosh for many valuable discussions and suggestions, and also thank for the anonymous reviewers for their helpful comments.

References

1 A. Abdulkadiroglu, P. Pathak, A. E. Roth, *Strategy-Proofness versus Efficiency in Matching with Indifferences: Redesigning the NYC High School Match*, American Economic Review, V.99(5), 1954-1978, 2009.

2 A. Abdulkadiroglu, T. Sonmez. *School Choice: A Mechanism Design Approach*, American Economic Review, V.93(3), 729-747, 2003.
3 N. Chen, A. Ghosh, *Strongly Stable Assignment*, ESA 2010, 147-158.
4 N. Chen, A. Ghosh, *A Market Clearing Solution for Social Lending*, IJCAI 2011, 152-157.
5 F. Echenique, J. Oviedo, *A Theory of Stability in Many-to-Many Matching Markets*, Theoretical Economics. V.1(2), 233-273, 2006.
6 A. Erdil, H. Ergin, *What's the Matter with Tie-breaking? Improving Efficiency in School Choice*, American Economic Review, V.98(3), 669-689, 2008.
7 A. Erdil, H. Ergin, *Two-Sided Matching with Indifferences*, working paper.
8 D. Gale, L. S. Shapley, *College Admissions and the Stability of Marriage*, American Mathematical Monthly, V.69, 9-15, 1962.
9 D. Gusfield, R. W. Irving, *The Stable Marriage Problem: Structure and Algorithms*, MIT Press, 1989.
10 T. Kavitha, K. Mehlhorn, D. Michail, K. E. Paluch, *Strongly Stable Matchings in Time O(nm) and Extension to the Hospitals-Residents Problem*, ACM Transactions on Algorithms, V.3(2), 2007.
11 K. Iwama, D. Manlove, S. Miyazaki, Y. Morita, *Stable Marriage with Incomplete Lists and Ties*, ICALP 1999, 443-452.
12 K. Iwama, S. Miyazaki, *Stable Marriage with Ties and Incomplete Lists*, Encyclopedia of Algorithms, 2008.
13 K. Iwama, S. Miyazaki, N. Yamauchi, *A 1.875-Approximation Algorithm for the Stable Marriage Problem*, SODA 2007, 288-297.
14 R. W. Irving, *Stable Marriage and Indifference*, Discrete Applied Mathematics, V.48, 261-272, 1994.
15 D. F. Manlove, *Stable Marriage with Ties and Unacceptable Partners*, Technical Report TR-1999-29, University of Glasgow, 1999.
16 D. F. Manlove, R. W. Irving, K. Iwama, S. Miyazaki, Y. Morita, *Hard Variants of Stable Marriage*, Theoretical Computer Science, V.276(1-2), 261-279, 2002.
17 E. McDermid, *A 3/2-Approximation Algorithm for General Stable Marriage*, ICALP 2009, 689-700.
18 A. E. Roth, *Deferred Acceptance Algorithms: History, Theory, Practice, and Open Questions*, International Journal of Game Theory, 537-569, 2008.
19 A. E. Roth, M. Sotomayor, *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis*, Cambridge University Press, 1992.
20 A. Roth, J. Vande Vate, *Random Paths to Stability in Two-Sided Matching*, Econometrica, V.58, 1475-1480, 1990.
21 M. Sotomayor, *Three Remarks on the Many-to-Many Stable Matching Problem*, Mathematical Social Sciences, V.38(1), 55-70, 1999.
22 M. Sotomayor, *The Pareto-Stability Concept is a Natural Solution Concept for Discrete Matching Markets with Indifferences*, International Journal of Game Theory, 2010.

On the separation question for tree languages

André Arnold[1], Henryk Michalewski*[2], and Damian Niwiński†[2,3]

1 **Talence,** andre.arnold@club-internet.fr
2 **University of Warsaw**
Faculty of Mathematics, Informatics, and Mechanics
{H.Michalewski,D.Niwinski}@mimuw.edu.pl
3 **Institute of Mathematics**
Polish Academy of Sciences

Abstract

We show that the separation property fails for the classes Σ_n of the Rabin-Mostowski index hierarchy of alternating automata on infinite trees. This extends our previous result (obtained with Szczepan Hummel) on the failure of the separation property for the class Σ_2 (i.e., for co-Büchi sets). It remains open whether the separation property does hold for the classes Π_n of the index hierarchy. To prove our result, we first consider the Rabin-Mostowski index hierarchy of deterministic automata on infinite words, for which we give a complete answer (generalizing previous results of Selivanov): the separation property holds for Π_n and fails for Σ_n-classes. The construction invented for words turns out to be useful for trees *via* a suitable game.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.1 Mathematical Logic, F.4.3 Formal Languages

Keywords and phrases Alternating automata on infinite trees, Index hierarchy, Separation property

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.396

1 Introduction

The separation question is whether two disjoint sets A and B can be separated by a set C (i.e., $A \subseteq C$ and $B \subseteq \overline{C}$) which is in some sense simpler. Separation is one of the main issues in descriptive set theory. A fundamental result due to Lusin is that two analytic sets can be always separated by a Borel set, but two co-analytic sets in general cannot. The former implies that if a set is simultaneously analytic and co-analytic then it is necessarily Borel, which is the celebrated Suslin Theorem (see, e.g., [8] or [7]).

A well-known fact in automata theory exhibits a similar pattern: if a set of infinite trees as well as its complement are both recognizable by Büchi automata then they are also recognizable by weak alternating automata (weakly recognizable, for short). This result was first proved by Rabin [10] in terms of monadic second-order logic; the automata-theoretic statement was given by Muller, Saoudi, and Schupp [9]. It is not difficult to adapt Rabin's proof to obtain the separation property: any two disjoint Büchi recognizable sets of trees can be separated by a weakly recognizable set (see, e.g., [5]). Quite analogical to the co-analytic case, the separation property fails in general for the dual class of co-Büchi tree languages

* Supported by the Polish Ministry of Science grant nr N N206 567840.
† Supported by the Polish Ministry of Science grant nr N N206 567840.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 396–407

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


(i.e., the complements of Büchi recognizable sets). In [5], a pair of such sets is presented that cannot be separated by any Borel set, hence *a fortiori* by any weakly recognizable set.

A systematic study of the separation property for tree automata has been undertaken by Santocanale and Arnold [11]. They asked if the above-mentioned result of Rabin can be shifted to the higher levels of the index hierarchy of alternating automata with an appropriate generalization of weak recognizability. The question stems naturally from the μ-calculus version of Rabin's result which states that if a tree language is definable both by a Π_2-term (i.e., with a pattern $\nu\mu$) and a Σ_2-term ($\mu\nu$), then it is also definable by an alternation free term, i.e., one in $\mathit{Comp}(\Pi_1 \cup \Sigma_1)$ [2]. Somewhat surprisingly, Santocanale and Arnold [11] discovered that the equation

$$\Pi_n \cap \Sigma_n = \mathit{Comp}(\Pi_{n-1} \cup \Sigma_{n-1}),$$

which amounts to Rabin's result for $n = 2$, fails for all $n \geq 3$. Consequently, it is in general not possible to separate two disjoint sets in the class Σ_n by a set in $\mathit{Comp}(\Pi_{n-1} \cup \Sigma_{n-1})$; similarly for Π_n.

There is however another plausible generalisation of Rabin's result suggested by the analogy with descriptive set theory. Letting

$$\Delta_n = \Pi_n \cap \Sigma_n,$$

we may ask if two disjoint sets in a class Σ_n can be separated by a set in Δ_n; a similar question can be stated for Π_n. By remarks above, we know that the separation property in this sense holds for Π_2 (Büchi) and fails for Σ_2 (co-Büchi) class, in a perfect analogy with the properties of analytic vs. co-analytic classes in the descriptive set theory[1].

In the present paper we answer the question negatively for all classes Σ_n of the Rabin–Mostowski index hierarchy for alternating automata on infinite trees. (The Σ_n-classes correspond to the indices (i, k) with k odd; see the definition below.) By an analogy with the Borel hierarchy [7, 8], one is tempted to conjecture that the separation property actually does hold for all classes Π_n, but this question seems to be difficult already for $n = 3$.

To prove our main result, we first study a conceptually simpler case of infinite words and the Rabin–Mostowski index hierarchy of deterministic automata. In this case we give a complete answer: the separation property holds for classes Π_n and fails for classes Σ_n. The argument is based on a uniform construction of an inseparable pair in each class Σ_n. This construction is further used in the case of trees. More specifically, we consider labeled trees whose vertices are divided between two players: Eve and Adam, who wish to form a path in the tree. For a set on infinite words L, we consider the set $\mathit{Win}^{\exists}(L)$ of those trees where Eve has a strategy to force a path into L. The operation $\mathit{Win}^{\exists}$ allows us to shift the witness family from words to trees.

It should be noted that in the case of deterministic automata on infinite words, the separation property of the class $(1, 2)$ was proved earlier by Selivanov [12], who also gave a hint [13] how this result can be generalized for all classes Π_n.

2 Index hierarchy

Throughout the paper, ω stands for the set of natural numbers, which we identify with its ordinal type. We also identify a natural number $n < \omega$ with the set $\{0, 1, \ldots, n-1\}$.

[1] However, the classical notation plays a trick here, as the analogy matches the classes $\mathbf{\Sigma}^1_1 \sim \Pi_2$ and $\mathbf{\Pi}^1_1 \sim \Sigma_2$.

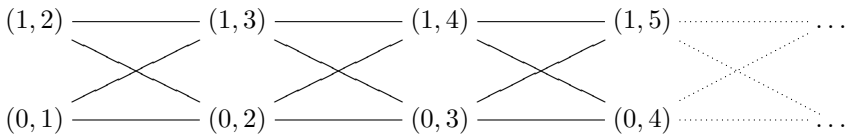


Figure 1 The Mostowski–Rabin index hierarchy.

We will consider deterministic automata on infinite words and alternating automata on infinite trees. For more background, we refer the reader to a survey by Thomas [14].

A *deterministic parity automaton* over an input alphabet A can be presented by $\mathcal{A} = \langle A, Q, q_I, Tr, rank\rangle$, where Q is a finite set of *states* ranked by the function $rank : Q \to \omega$, and $Tr : Q \times A \to Q$ is a *transition* function. A *run* of $\mathcal{A}$ on a word $u \in A^\omega$ is a word $r \in Q^\omega$ whose first element r_0 is the *initial state* q_I, and $r_{n+1} = Tr(r_n, u_n)$, for $n < \omega$. It is *accepting* if the highest rank occurring infinitely often (i.e., $\limsup_{n\to\infty} rank(r_n)$) is *even*. The language $L(\mathcal{A})$ recognized by $\mathcal{A}$ consists of those words $u \in A^\omega$ which admit an accepting run. The *Rabin-Mostowski index* of $\mathcal{A}$ is the pair $(\min rank(Q), \max rank(Q))$; we may assume without loss of generality that $\min rank(Q)$ is 0 or 1. It is useful to partially order the indices as represented on Figure 1. That is, we let $(\iota, \kappa) \sqsubseteq (\iota', \kappa')$ if either $\{\iota, \dots, \kappa\} \subseteq \{\iota', \dots, \kappa'\}$, or $\iota = 0$, $\iota' = 1$, and $\{\iota+2, \dots, \kappa+2\} \subseteq \{\iota', \dots, \kappa'\}$. We consider the indices $(1, \kappa)$ and $(0, \kappa-1)$ as *dual*, and let $\overline{(\iota, \kappa)}$ denote the index dual to (ι, κ). The above ordering induces a hierarchy, that is, if a language L is recognized by an automaton of index (ι, κ) and $(\iota, \kappa) \sqsubseteq (\iota', \kappa')$ then L is also recognized by an automaton of index (ι', κ'). Moreover, the hierarchy is *strict* in the sense that, for any index (ι, κ), there is a language recognized by an automaton of index (ι, κ), but not by any (deterministic) automaton of the dual index $\overline{(\iota, \kappa)}$ [6, 15]. Indeed, the witness can be the parity condition itself:

$$L_{\iota,\kappa} \quad = \quad \{u \in \{\iota, \dots, \kappa\}^\omega : \limsup_{n\to\infty} u_n \text{ is even } \}. \tag{1}$$

The concept of alternating automaton is best understood via parity games. For the sake of further application, we present them in a more general setting of *graph games*. A graph game is a perfect information game of two players, say Eve and Adam, where plays may have infinite duration. It can be presented by a tuple

$$\langle V_\exists, V_\forall, Move, p_I, \ell, A, L_\exists, L_\forall\rangle.$$

Here $V_\exists$ and $V_\forall$ are (disjoint) sets of positions of Eve and Adam, respectively, $Move \subseteq V \times V$ is the relation of possible moves, with $V = V_\exists \cup V_\forall$, $p_I \in V$ is a designated initial position, and $\ell : V \to A$ is a labelling function, with some alphabet A. These items constitute an arena of the game. Additionally, $L_\exists, L_\forall \subseteq A^\omega$ are two disjoint sets representing the *winning criteria* for Eve and Adam, respectively.

The players start a play in the position p_I and then move the token according to the relation *Move* (always to a successor of the current position), thus forming a path in the arena. The move is selected by Eve or Adam, depending on who the owner of the current position is. If a player cannot move, she/he looses. Otherwise, the result of the play is an infinite path $v_0, v_1, v_2, \dots$, inducing the sequence of labels $\ell(v_0), \ell(v_1), \ell(v_2), \dots$ If this sequence belongs to $L_\exists$ then Eve wins, if it belongs to $L_\forall$ then Adam wins; otherwise there is a draw. We say that Eve *wins* the game if she has a winning strategy, the similar for Adam.

In the games considered in this paper, we always have $L_\forall = A^\omega - L_\exists$, hence a draw will not occur. But it is convenient to consider the winning criteria for both players.

A *parity game of index* (ι, κ) is defined as above with $A = \{\iota, \ldots, \kappa\}$, $L_\exists = L_{\iota,\kappa}$ (see equation (1)), and $L_\forall = \overline{L_\exists} = \{u \in \{\iota, \ldots, \kappa\}^\omega : \limsup_{n\to\infty} u_n \text{ is odd }\}$.

A (full) *k-ary tree* over a finite alphabet A is a mapping $t : k^* \to A$. An *alternating parity tree automaton of index* (ι, κ) running on such trees can be presented by

$$\mathcal{A} = \langle A, Q_\exists, Q_\forall, q_I, \delta, \mathrm{rank}\rangle$$

where Q is a finite set of states with an initial state q_I, partitioned into existential states $Q_\exists$ and universal states $Q_\forall$, $\delta \subseteq Q \times A \times \{0, 1, \ldots, k-1, \varepsilon\} \times Q$ is a transition relation, and $rank : Q \to \omega$. An input tree t is accepted by $\mathcal{A}$ iff Eve has a winning strategy in the parity game

$$G(\mathcal{A}, t) \quad = \quad \langle Q_\exists \times k^*, Q_\forall \times k^*, (q_0, \varepsilon), \mathrm{Mov}, \ell, L_{\iota,\kappa}, \overline{L_{\iota,\kappa}}\rangle, \tag{2}$$

where $\mathrm{Mov} = \{((p, v), (q, vd)) : v \in \mathrm{dom}(t),\ (p, t(v), d, q) \in \delta\}$ and $\ell(q, v) = \mathrm{rank}(q)$. Intuitively, the players follow a path in the tree t, additionally annotated by the states. A transition is always selected by the owner of the state. The automaton accepts the tree if Eve can force each such path to be accepting.

The hierarchy of tree languages induced by the indices of alternating parity automata is strict, as showed by Bradfield [4]. An alternative proof of this difficult result was later given [1] based on the Banach Fixed-Point Theorem; Both proofs [1, 4] use the same witness family of sets of binary trees defined by parity games. For the sake of further application, we will present this concept in a more general setting.

We consider k-ary trees over an alphabet $\{\exists, \forall\} \times A$. The labels $\{\exists, \forall\}$ are used to partition the nodes of a tree t into positions of Eve and Adam. In the game, the players form a path in t, starting from the root. The next move is selected by Eve or Adam, depending on whether the actual label contains $\exists$ or $\forall$. The winning criteria concern the sequence formed by the second components of the labels occurring in the play, which is a word in A^ω .

Each language $L \subseteq A^\omega$ induces two winning criteria: $L_\exists = L$, and $L_\forall = L$, which give rise to two games: an L-$\exists$ *game*, and an L-$\forall$ *game*. Let us describe formally an L-$\exists$ game over a tree $t : k^* \to \{\exists, \forall\} \times A$. It is a graph game with the following items:

$$\begin{array}{lcllcl} V_\exists & = & \{v \in k^* : t(v)\downarrow_1 = \exists\} & \ell(v) & = & t(v)\downarrow_2, \text{ for } v \in k^* \\ V_\forall & = & \{v \in k^* : t(v)\downarrow_1 = \forall\} & L_\exists & = & L \\ Move & = & \{(w, wi) : w \in k^*, i \in k\} & L_\forall & = & \overline{L}. \\ p_0 & = & \varepsilon \text{ (the root of the tree)} & & & \end{array}$$

An L-$\forall$ game is defined similarly with the winning criteria $L_\forall = L$, and $L_\exists = \overline{L}$.

The set $Win_k^\exists(L)$ consists of those trees t, for which Eve has a winning strategy in L-$\exists$-game. The set $Win_k^\forall(L)$ consists of those trees t, for which Adam has a winning strategy in L-$\forall$-game. The following can be easily verified.

▶ Fact 1. If a language L of infinite words is recognized by a deterministic automaton of index (ι, κ) then both languages $Win_k^\exists(L)$ and $Win_k^\forall(L)$ can be recognized by an alternating[2] tree automaton of index (ι, κ).

[2] In fact, even *non-deterministic*, but we don't explore it in this paper.

A family witnessing the strictness of the index hierarchy of alternating tree automata [1, 4] consists of the sets of binary trees $W_{i,k}$, which can be presented by $W_{i,k} = Win_2^{\exists}(L_{i,k})$.

The following Σ/Π terminology for the index hierarchy,motivated by the connection with the μ-calculus (see, e.g., [3]); will be convenient to handle dualities. For each $m \geq 1$, we consider two indices: $(1, m)$ and $(0, m-1)$, and associate the symbol Σ_m with this index whose maximum is odd, and Π_m with the one whose maximum is even. For example, $(0,1) \approx \Sigma_2$, $(1,2) \approx \Pi_2$, $(1,3) \approx \Sigma_3$, $(0,2) \approx \Pi_3$, $(1,4) \approx \Pi_4$, $(0,3) \approx \Sigma_4$, etc. We will then refer to an automaton of index (ι, κ) as to Σ_m or Π_m-automaton with an appropriate m.

3 Deterministic hierarchy over words

In this section we investigate the index hierarchy for deterministic automata on infinite words. A language of infinite words is in the class Σ_m if it is recognized by a deterministic Σ_m-automaton; similarly for Π_m. A language is in the class Δ_m if it is simultaneously Σ_m and Π_m. We show that the separation property holds for classes Π_m and fails for Σ_m, for $m \geq 2$. In fact, both properties will follow from a single construction (parametrized by m).

Note that we do not consider the classes Σ_1 and Π_1, which are uninteresting from the point of view of the separation property[3].

For $m \geq 2$, we fix an alphabet

$$m_\pi = \begin{cases} \{1, \ldots, m\} & \text{if } m \text{ is even} \\ \{0, \ldots, m-1\} & \text{if } m \text{ is odd.} \end{cases}$$

Note that $\max m_\pi$ is always even. Let $I_m \subseteq m_\pi^\omega$ be the set of infinite words where $\max m_\pi$ occurs infinitely often. Let K_m be a superset of I_m consisting of the words satisfying parity condition,

$$K_m \quad = \quad \{u \in m_\pi^\omega : \limsup_{n \to \infty} u_n \text{ is even } \}.$$

That is, K_m coincides with $L_{1,m}$ or $L_{0,m-1}$ of (1), depending on whether m is even or odd. It is straightforward to see that K_m is in the class Π_m.

In the sequel we consider words over a product alphabet m_π^2. We identify a pair of words $\langle u, v \rangle \in (m_\pi^\omega)^2$ with a single word over $\left(m_\pi^2\right)$ in an obvious manner. The subsequent lemma is the heart of our paper.

▶ **Lemma 1.** *For each $m \geq 2$, there exist disjoint sets $U_1, U_2 \subseteq m_\pi^\omega$ of class Σ_m, satisfying the following:*

$$\begin{array}{lcl} K_m \times \overline{K_m} & \subseteq & U_1 \\ \overline{K_m} \times K_m & \subseteq & U_2 \\ \overline{K_m \times K_m} & \subseteq & U_1 \cup U_2 \quad = \quad \overline{I_m \times I_m}. \end{array}$$

Proof. We first present the construction in the case when m is odd; thus the Σ_m-automata have index $(1, m)$, and Π_m-automata have index $(0, m-1)$.

Let P_m be an automaton over the alphabet m_π with the set of states also equal to m_π, and the transition function $Tr(q, s) = s$, for any q and s. (We leave the remaining items temporarily unspecified.) Let $P_m \times \top$ be an automaton over m_π^2 which behaves like P_m reading only the first component. Similarly, $\top \times P_m$ reads only the second component.

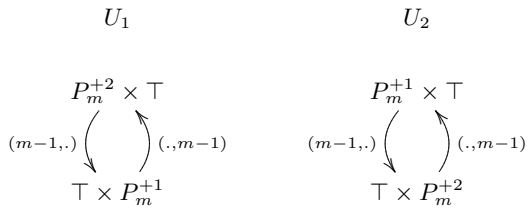


Figure 2 Automata for U_1 and U_2 for m odd.

We represent the Σ_m-automata recognizing U_1 and U_2 on Figure 2. The states of the automaton for U_1 are $\{0, 1, \ldots, m-1\}$ (upper component) and $\{0', 1', \ldots, (m-1)'\}$ (lower component). In its upper component, the automaton reads the left component of the input symbols until it eventually encounters a symbol $(m-1, s)$, for some s. Then the edge is directed to the state $(m-1)'$ in the lower component. Here the automaton reads the right component of the input symbols until it eventually encounters a symbol $(s, m-1)$, for some s, in which case it moves to the state $m-1$ in the upper component. The ranks in the upper component are $\mathit{rank}(i) = i+2$, for $i = 0, 1, \ldots, m-2$, and $\mathit{rank}(m-1) = m$. The ranks in the lower component are $\mathit{rank}(i') = i+1$, for all i. For the initial state we set 0.

The automaton for U_2 is defined analogously; the difference concerns only rankings[4], namely $\mathit{rank}(i) = i+1$, for all i, and $\mathit{rank}(i') = i+2$, for $i = 0, 1, \ldots, m-2$ and $\mathit{rank}((m-1)') = m$. Note that the states $m-1$ and $(m-1)'$ can be reached only while changing the levels and, whenever it happens, both automata assume the highest odd rank m.

Each word $u \in \left(m_\pi^2\right)^\omega$ induces the same run in both automata up to the rankings. Clearly, a word u causes infinitely many changes of the level if and only if it contains infinitely many occurrences of $m-1$ on both left and right track. By remark above, such a word is accepted by neither of the automata. On the other hand, if the run on u stabilizes on some level then one of the automata necessarily accepts, as the ranks they assume in their runs (after stabilization) differ precisely by 1.

This shows that U_1 and U_2 are disjoint and $U_1 \cup U_2 = \overline{I_m \times I_m}$. The inclusion $\overline{K_m \times K_m} \subseteq \overline{I_m \times I_m}$ is obvious.

If $u \in K_m \times \overline{K_m}$ then the run on u stabilizes in the upper or lower component (as clearly $u \notin I_m \times I_m$). Then the automaton for U_1, from some moment on, either reads a word in K_m in the upper component or a word in $\overline{K_m}$ in the lower component; in either case it accepts. A similar argument shows the second inclusion, which completes the proof of the lemma in case m is odd. The construction for the case of m even is analogous. We leave it to the reader with Figure 3 as a hint.

◀

The properties of U_1 and U_2 mentioned in Lemma 1 imply a kind of hardness of these sets. Generally, for an automaton $\mathcal{A}$ over an alphabet A of some index (ι, κ), let $\mathit{rank}^{\mathcal{A}}$ denote the function sending a word $u \in A^\omega$ onto the sequence of ranks assumed by $\mathcal{A}$. More precisely,

[3] By definition, a deterministic automaton of index $(1,1)$ accepts no words, and an automaton of index $(0,0)$ accepts all words.

[4] The somewhat awkward exceptions in ranking of $(m-1)$ in the first automaton and $(m-1)'$ in the second follow from our desire of having the graphs of both automata the same. Otherwise we could merge the "nasty" states with their companions.

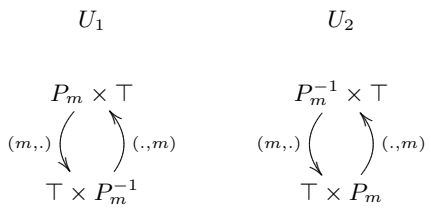


Figure 3 Automata for U_1 and U_2 for m even.

$rank^{\mathcal{A}} : A^\omega \to \{\iota, \ldots, \kappa\}^\omega$, and

$$rank^{\mathcal{A}}(u) \quad = \quad rank(r_0)rank(r_1)\ldots, \tag{3}$$

where $r_0 r_1 \ldots$ is the unique run of $\mathcal{A}$ on u. If $\mathcal{B}$ is another automaton over A with index (ι, κ), we define $rank^{\mathcal{A}\times\mathcal{B}} : A^\omega \to \left(\{\iota, \ldots, \kappa\}^2\right)^\omega$, by

$$rank^{\mathcal{A}\times\mathcal{B}}(u) \quad = \quad \langle rank^{\mathcal{A}}(u), rank^{\mathcal{B}}(u)\rangle.$$

▶ **Lemma 2.** *Let $\mathcal{A}$ and $\mathcal{B}$ be automata of class Π_m over some alphabet A, such that $L(\mathcal{A}) \cap L(\mathcal{B}) = \emptyset$. Let U_1 and U_2 satisfy the properties of Lemma 1. Then, for each word $u \in A^\omega$,*

1. *if $u \in L(\mathcal{A})$ then $rank^{\mathcal{A}\times\mathcal{B}}(u) \in U_1$,*
2. *if $u \in L(\mathcal{B})$ then $rank^{\mathcal{A}\times\mathcal{B}}(u) \in U_2$,*
3. *$rank^{\mathcal{A}\times\mathcal{B}}(u) \in U_1 \cup U_2$.*

Proof. Generally, if $\mathcal{D}$ is a deterministic parity automaton of index (ι, κ) then, by definition of acceptance,

$$u \in L(\mathcal{D}) \quad \Leftrightarrow \quad rank^{\mathcal{D}}(u) \in L_{\iota,\kappa}.$$

Hence, in our case, $u \in L(\mathcal{A}) \Rightarrow rank^{\mathcal{A}}(u) \in K_m$, and $u \notin L(\mathcal{B}) \Rightarrow rank^{\mathcal{B}}(u) \in \overline{K_m}$. As $L(\mathcal{A})$ and $L(\mathcal{B})$ are disjoint, $u \in L(\mathcal{A})$ implies $rank^{\mathcal{A}\times\mathcal{B}}(u) \in K_m \times \overline{K_m}$, but we know from Lemma 1 that $K_m \times \overline{K_m} \subseteq U_1$. The argument for 2 is similar. Finally, again by disjointness of $L(\mathcal{A})$ and $L(\mathcal{B})$, we have $rank^{\mathcal{A}\times\mathcal{B}}(u) \in \overline{K_m \times K_m}$, for any u, but we know from Lemma 1 that $\overline{K_m \times K_m} \subseteq U_1 \cup U_2$, which completes the proof. ◀

We are now ready to state the main result of this section. Recall that the separation property for the class Π_2 was proved earlier by Selivanov [12], who also gave[5] a hint [13] how this result can be generalized for all classes Π_n.

▶ **Theorem 3.** *The separation property holds for classes Π_m and fails for classes Σ_m of the index hierarchy of deterministic word automata.*

Proof. We will show that any pair of disjoint languages of class Π_m over some finite alphabet A is separable by a language of class Δ_m, whereas this property fails for the pair of sets U_1, U_2 of Lemma 1 (which are of class Σ_m).

Let $\mathcal{A}$ and $\mathcal{B}$ be as in Lemma 2. It follows from 1 and 2 that the inverse image of U_1 under the mapping $rank^{\mathcal{A}\times\mathcal{B}}$, i.e.,

$$\left(rank^{\mathcal{A}\times\mathcal{B}}\right)^{-1}(U_1) \quad = \quad \{u \in A^\omega : rank^{\mathcal{A}\times\mathcal{B}}(u) \in U_1\}$$

[5] More precisely, that author considered the *reduction* property for the dual classes Σ_m. See a comment after Proposition 3.5 in [13].

separates $L(\mathcal{A})$ and $L(\mathcal{B})$. Let us see that this set is recognizable by an Σ_m-automaton. For an input u, we just use the automaton for U_1 reading the subsequent values of the function $rank^{\mathcal{A}\times\mathcal{B}}$; the construction is straightforward. In a similar vein we can show that $\left(rank^{\mathcal{A}\times\mathcal{B}}\right)^{-1}(U_2)$ is in the class Σ_m as well. Clearly these sets are disjoint as U_1 and U_2 are disjoint. But it follows from condition 3 of Lemma 2 that they sum up to A^ω, hence they both are of class Δ_m.

To show that U_1 and U_2 are inseparable, we start with the following observation. Suppose that $\mathcal{A}$ and $\mathcal{B}$ are Π_m-automata over the alphabet m_π^2. Then the function $rank^{\mathcal{A}\times\mathcal{B}}$, which in this case has type $rank^{\mathcal{A}\times\mathcal{B}} : \left(m_\pi^2\right)^\omega \to \left(m_\pi^2\right)^\omega$, has a fixed point. Indeed, a fixed point f can be defined[6] by an inductive formula

$$\begin{aligned} f_0 &= \left(rank(q_I^{\mathcal{A}}), rank(q_I^{\mathcal{B}})\right) \\ f_{n+1} &= \left(rank\left(\hat{Tr}^{\mathcal{A}}(q_I^{\mathcal{A}}, f_0 \dots f_n)\right), rank\left(\hat{Tr}^{\mathcal{B}}(q_I^{\mathcal{B}}, f_0 \dots f_n)\right)\right) \end{aligned}$$

(where $\hat{Tr}$ is the standard extension of Tr from letters to finite words).

Now suppose, for the sake of contradiction, that there is a set $C \subseteq m_\pi^\omega$ of class Δ_m, such that $U_1 \subseteq C$ and $U_2 \subseteq \overline{C}$. Let $\mathcal{A}$ and $\mathcal{B}$ be two automata of class Π_m, such that

$$\begin{aligned} L(\mathcal{A}) &= \overline{C} \\ L(\mathcal{B}) &= C. \end{aligned}$$

By remark above, the function $rank^{\mathcal{A}\times\mathcal{B}}$ has the fixed point $(f_0, f_1, \dots)$. On the other hand, by conditions 1 and 2 of Lemma 2, it reduces $\overline{C}$ to $U_1 \subseteq C$, and C to $U_2 \subseteq \overline{C}$. This contradiction completes the proof. ◀

4 Alternating hierarchy over trees

In this section we investigate the Rabin-Mostowski index hierarchy for alternating automata on k-ary trees. A tree language is in the class Σ_m if it is recognized by an alternating Σ_m-automaton; similarly for Π_m. A tree language is in the class Δ_m if it is simultaneously Σ_m and Π_m. We show that the separation property fails in general for the classes Σ_m, for $m \geq 1$.

It will be convenient to have some normal form of alternating automata. We call an alternating automaton on k-ary trees, $\mathcal{A} = \langle A, Q_\exists, Q_\forall, q_I, \delta, \mathrm{rank}\rangle$, an $\exists\forall$-automaton if it satisfies the following conditions:

1. $q_I \in Q_\exists$,
2. if $(p, s, d, q) \in \delta$ is a transition then $p \in Q_\exists$ iff $q \in Q_\forall$,
3. for any pair $(p, s) \in Q \times A$, there are *exactly* two transitions $(p, s, d, q), (p, s, d', q') \in \delta$.

These conditions imply that the graph of the game $G(\mathcal{A}, t)$ (see equation (2)) unravels to a full binary tree, where $\exists$ and $\forall$ alternate starting with $\exists$.

We will focus on the ranks of the states occurring in this tree. More precisely, let the index of $\mathcal{A}$ be (ι, κ). With any tree $t : k^* \to A$, we associate a binary tree

$$\mathcal{T}(\mathcal{A}, t) : 2^* \to \{\exists, \forall\} \times \{\iota, \dots, \kappa\}$$

[6] The existence and uniqueness of this fixed point can be also inferred from the Banach Fixed-Point Theorem.

as follows. We define an auxiliary function $\gamma : 2^* \to Q \times k^*$, using the notation $own(q) = \xi \in \{\exists, \forall\}$, whenever $q \in Q_\xi$. The value of γ represents the state and the current position in the game-play on the tree t. Let $\gamma(\varepsilon) = (q_I, \varepsilon)$, and $\mathcal{T}(\mathcal{A}, t)((\varepsilon) = (own(q_I), rank(q_I))$. Suppose $\mathcal{T}(\mathcal{A}, t)(v)$ and $\gamma(v)$ are defined, say $\gamma(v) = (p, w)$. By condition 3 above, there are exactly two pairs that extend $(p, t(w))$ to a transition in δ. Suppose they are (d, q) and (d', q') (in this pre-defined order, for definiteness). We let $\gamma(v0) = (q, wd)$ and $\gamma(v1) = (q', wd')$. We further define $\mathcal{T}(\mathcal{A}, t)(v0) = (own(q), rank(q))$ and $\mathcal{T}(\mathcal{A}, t)(v1) = (own(q'), rank(q'))$. It is straightforward to see that

$$t \in L(\mathcal{A}) \quad \Leftrightarrow \quad \mathcal{T}(\mathcal{A}, t) \in W_{\iota,\kappa} \tag{4}$$

(see page 400 for the definition of $W_{\iota,\kappa}$). We leave to the reader the proof of the following simple observation.

▶ **Lemma 4.** *Any alternating tree automaton can be transformed to an $\exists\forall$-automaton of the same index, recognizing the same language.*

A $\forall\exists$-automaton is defined similarly, with the only difference that $q_I \in Q_\forall$. Clearly, an analogue of Lemma 4 for $\forall\exists$-automata holds as well.

We are going to define a tree version of the "hard pairs" from Section 3. Let $m \geq 1$, and let U_1 and U_2 be the languages defined in the proof of Lemma 1. We let

$$\begin{aligned} \nabla_1 &= Win_4^\exists(U_1) \\ \nabla_2 &= Win_4^\forall(U_2). \end{aligned}$$

By Fact 1, the sets ∇_1 and ∇_2 are of class Σ_m. To have some analogue of Lemma 2, we need some analogue of the function $rank^{\mathcal{A}\times\mathcal{B}}$; we will define it only for automata in a special form.

We call a tree $t : k^* \to \{\exists, \forall\} \times A$ a $\exists\forall$-tree if

$$t(v)\downarrow_1 = \begin{cases} \exists & \text{if } |v| \text{ is even} \\ \forall & \text{if } |v| \text{ is odd.} \end{cases}$$

The concept of a $\forall\exists$-tree is defined analogously. Note that the trees $\mathcal{T}(\mathcal{A}, t)$ defined above are $\exists\forall$-trees or $\forall\exists$-trees, whenever $\mathcal{A}$ is an $\exists\forall$-automaton or $\forall\exists$-automaton, respectively.

At first, we define a *product* of a binary $\exists\forall$-tree t_1 and a binary $\forall\exists$ tree t_2 as a 4-ary $\exists\forall$-tree $t_1 \star t_2 : 4^* \to \{\exists, \forall\} \times A \times A$. It is convenient to fix some bijection $4 \sim 2 \times 2$, so that a (finite) word w in 4^* can be identified with a pair of words v, u in 2^* of the same length (such that $w_i = (v_i, u_i)$); we then use the notation $w = (u \circ v)$. We then let[7]

$$t_1 \star t_2(u \circ v) \quad = \quad (t_1(u)\downarrow_1, t_1(u)\downarrow_2, t_2(v)\downarrow_2)\,.$$

Now fix k and an alphabet A. Let $\mathcal{A}$ and $\mathcal{B}$ be two automata on k-ary trees over the alphabet A, both of the class Π_m. Assume moreover that $\mathcal{A}$ is an $\exists\forall$-automaton and $\mathcal{B}$ a $\forall\exists$-automaton. For a tree $t : k^* \to A$, consider an $\exists\forall$-tree $\mathcal{T}(\mathcal{A}, t)$ and a $\forall\exists$-tree $\mathcal{T}(\mathcal{B}, t)$. We let

$$g^{\mathcal{A}\times\mathcal{B}}(t) \quad = \quad (\mathcal{T}(\mathcal{A}, t) \star \mathcal{T}(\mathcal{B}, t)). \tag{5}$$

The following is a (partial) analogue of Lemma 2.

[7] Figure 4 at the end of the paper shows how from a green $\exists\forall$ tree t_1 and a red $\forall\exists$ tree t_2 we obtain a blue 4-ary tree $t_1 \star t_2$.

▶ **Lemma 5.** *With the notations above,*
1. *if* $t \in L(\mathcal{A})$ *then* $g^{\mathcal{A}\times\mathcal{B}}(t) \in \nabla_1$,
2. *if* $t \in L(\mathcal{B})$ *then* $g^{\mathcal{A}\times\mathcal{B}}(t) \in \nabla_2$.

Proof. Assume that $t \in L(\mathcal{A})$. It implies, that Eve has a winning strategy $\sigma_{\mathcal{A}}$ showing that $\mathcal{T}(\mathcal{A}, t) \in W_{\iota,\kappa}$. Since $L(\mathcal{A})$ and $L(\mathcal{B})$ are disjoint, Adam has a winning strategy $\sigma_{\mathcal{B}}$ showing that $\mathcal{T}(\mathcal{B}, t) \notin W_{\iota,\kappa}$.

Let σ be the strategy for Eve on the tree $\mathcal{T}(\mathcal{A}, t) \star \mathcal{T}(\mathcal{B}, t)$ which combines $\sigma_{\mathcal{A}}$ and $\sigma_{\mathcal{B}}$. Namely, $\sigma_{\mathcal{A}}$, $\sigma_{\mathcal{B}}$ choose one of the two options available for respectively Eve and Adam and σ chooses the uniquely defined combination of these two options. Since $\sigma_{\mathcal{A}}$ leads to a sequence in K_m, $\sigma_{\mathcal{B}}$ leads to a sequence in the complement of K_m, the resulting sequence defined by σ belongs to $K_m \times \overline{K_m}$, in particular it belongs to U_1.

Assume now that $t \in L(\mathcal{B})$. It implies, that Eve has a winning strategy $\sigma_{\mathcal{B}}$ showing that $\mathcal{T}(\mathcal{B}, t) \in W_{\iota,\kappa}$. Since $L(\mathcal{A})$ and $L(\mathcal{B})$ are disjoint, Adam has a winning strategy $\sigma_{\mathcal{A}}$ showing that $\mathcal{T}(\mathcal{A}, t) \notin W_{\iota,\kappa}$.

Let σ be the strategy for Adam on the tree $\mathcal{T}(\mathcal{A}, t) \star \mathcal{T}(\mathcal{B}, t)$ which combines $\sigma_{\mathcal{A}}$ and $\sigma_{\mathcal{B}}$. Namely, $\sigma_{\mathcal{A}}$, $\sigma_{\mathcal{B}}$ choose one of the two options available for respectively Adam and Eve and σ chooses the uniquely defined combination of these two options. Since $\sigma_{\mathcal{A}}$ leads to a sequence in the complement of K_m, $\sigma_{\mathcal{B}}$ leads to a sequence in K_m, the resulting sequence defined by σ belongs to $\overline{K_m} \times K_m$, in particular it belongs to U_2. ◀

The reader may have noticed that the point 3 of Lemma 2 is missing in Lemma 5. This is precisely why we fail to extend the positive results on the classes Π_m from words to trees.

We can state the main result of the paper.

▶ **Theorem 6.** *The separation property fails for classes* Σ_m *of the index hierarchy of alternating tree automata. More specifically, for any* $m \geq 2$*, there exists a pair of sets of 4-ary trees of class* Σ_m *inseparable by any set of class* Δ_m.

Proof. The proof is similar to the proof of Theorem 3. To show that ∇_1 and ∇_2 are inseparable, we start with the following observation. Suppose that $\mathcal{A}$ and $\mathcal{B}$ are Π_m-automata over the alphabet $\{\exists, \forall\} \times (\iota, \kappa)^2$. Suppose moreover that $\mathcal{A}$ is an $\exists\forall$-automaton and $\mathcal{B}$ is a $\forall\exists$-automaton. We will show that the mapping $g^{\mathcal{A}\times\mathcal{B}}$ has a fixed point t. Since the range of the mapping $g^{\mathcal{A}\times\mathcal{B}}$ consists of $\exists\forall$ trees, it implies the first coordinate of $t(u \circ v)$ has to be $\exists$ for $u \circ v$ of even length and $\forall$ otherwise. The second and third coordinates of $t(u \circ v)$ for $u = u_0 \dots u_{n+1}$ and $v = v_0 \dots v_{n+1}$ will be defined along the same lines as in the proof of Theorem 3, however we have to take care of ϵ-transitions. For this sake we will define the token mappings $\gamma_{\mathcal{A}}$ and $\gamma_{\mathcal{B}}$ like the token mapping γ used in the definition of $\mathcal{T}(\mathcal{A}, t)$ (see (4)). The mappings will be defined successively together with the tree t.

Let us define

$$t(\varepsilon) \downarrow_2 = \mathit{rank}(q_I^{\mathcal{A}}), t(\varepsilon) \downarrow_3 = \mathit{rank}(q_I^{\mathcal{B}}).$$

The $\mathcal{A}$- and $\mathcal{B}$- tokens are placed in the root. In automaton $\mathcal{A}$ there are two transitions from q_I on the letter $t(\varepsilon)$. Similarly, there are two transitions in $\mathcal{B}$. The root of t has four successors uniquely defined by these two pairs of transitions. Assume now that Eve in $\mathcal{A}$ and Adam in $\mathcal{B}$ made their first moves. These two moves uniquely define a vertex $u_0 \circ v_0$ in the tree t, that is one of the four successors of the root of t. If Eve decided for an ϵ-transition then the second coordinate of $\gamma_{\mathcal{A}}$ remains unchanged. Otherwise we move the token to $u_0 \circ v_0$. Similarly if Adam decided for an ϵ-transition then the second coordinate of $\gamma_{\mathcal{B}}$ remains unchanged, otherwise we move the token to $u_0 \circ v_0$. In automaton $\mathcal{A}$ there are two

transitions from the state $\gamma_{\mathcal{A}}(u_0)\downarrow_1$ on the letter $t(\gamma_{\mathcal{A}}(u_0)\downarrow_2)$. Similarly, there are two transitions in $\mathcal{B}$ from the state $\gamma_{\mathcal{A}}(v_0)\downarrow_1$ on the letter $t(\gamma_{\mathcal{B}}(v_0)\downarrow_2)$. The vertex $t(u_0 \circ v_0)$ has four successors uniquely defined by these two pairs of transitions. We extend $\gamma_{\mathcal{A}}$ and γ_B accordingly and continue building a full 4-ary tree t.

It is easy to verify that the tree t is a fixed point[8] of $g^{\mathcal{A}\times\mathcal{B}}$. Now suppose, for the sake of contradiction, that there exists a set C of trees over the alphabet $\{\exists,\forall\}\times(\iota,\kappa)^2$ such that

- C belongs to the class Δ_m,
- $\nabla_1 \subseteq C$ and $\nabla_2 \subseteq \overline{C}$.

Let $\mathcal{A}$ and $\mathcal{B}$ be two automata of class Π_m, such that

$$L(\mathcal{A}) = \overline{C}$$
$$L(\mathcal{B}) = C.$$

By the remark above, the function $g^{\mathcal{A}\times\mathcal{B}}$ has the fixed point t. On the other hand, by conditions 1 and 2 of Lemma 5, it reduces $\overline{C}$ to $\nabla_1 \subseteq C$, and C to $\nabla_2 \subseteq \overline{C}$. This contradiction completes the proof. ◀

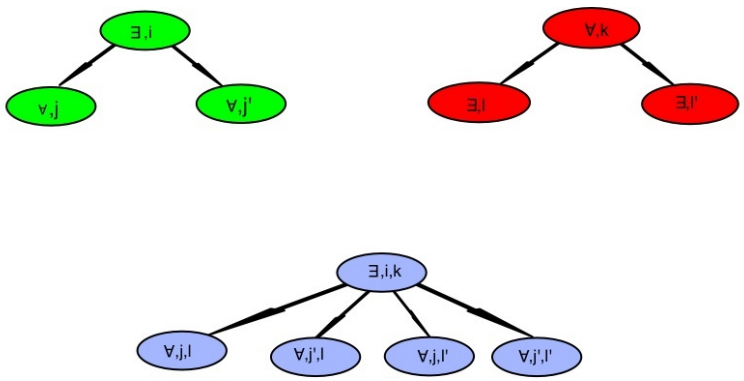


Figure 4 Operation $\star$ on a green $\exists\forall$ tree t_1 and a red $\forall\exists$ tree t_2 gives the blue 4-ary tree.

The consideration of 4-ary trees in Theorem 6 made the proof more transparent, but the result can be adapted to binary trees as well.

▶ **Corollary 7.** *There exists a pair of sets of binary trees of class Σ_m inseparable by any set of class Δ_m.*

Sketch of proof. Let ∇_1, ∇_2 be as in Theorem 6. We define languages V_1, V_2 of binary trees and a mapping η such that $\nabla_i = \eta^{-1}[V_i]$. Let V_1 consist of binary trees t over the alphabet $\{\exists,\forall\}\times m_\pi$ such that

1. the first two levels of t, that is the root and its children, belong to Eve, as well as all the levels $4k$, $4k+1$, for $k = 0, 1, 2, \ldots$,
2. the levels $4k+2$, $4k+3$, for $k = 0, 1, 2, \ldots$, belong to Adam,
3. Eve has a strategy, such that if the sequence $(\exists, w_0), (\exists, v_0), (\forall, w_1), (\forall, v_1), (\exists, w_2), (\exists, v_2), \ldots$ represents a game-play then $(w_0, v_0), (w_1, v_1), (w_2, v_2), \ldots$ belongs to U_1.

[8] As in Theorem 3, the existence and uniqueness of this fixed point can be also inferred from the Banach Fixed-Point Theorem.

The set V_2 satisfies the same conditions 1, 2, and 3 is replaced by the requirement that Adam has a strategy, which forces represents a game-play $(w_0, v_0), (w_1, v_1), (w_2, v_2), \ldots$ into U_2.

To a 4-ary tree t over the alphabet $\{\exists, \forall\} \times (m_\pi)^2$, we now assign a binary tree $t' = \eta(t)$ over the alphabet $\{\exists, \forall\} \times m_\pi$. As before, it is convenient to use a bijection $4 \sim 2 \times 2$, so that a node of t of level ℓ can be presented by $(x_0, y_0)(x_1, y_1), \ldots, (x_{\ell-1}, y_{\ell-1})$, with $x_i, y_i \in 2$. The root of t' is labeled $(t(\epsilon) \downarrow 1, t(\epsilon) \downarrow 2)$, and the second level is labeled by $(t(\epsilon) \downarrow 1, t(\epsilon) \downarrow 3)$ in both directions. More specifically, whenever

$$t\,((x_0, y_0)(x_1, y_1), \ldots (x_{\ell-1}, y_{\ell-1})) \quad = \quad (\xi, a, b),$$

we let

$$\begin{array}{llll} t' & (x_0, y_0, x_1, y_1, \ldots, x_{\ell-1}, y_{\ell-1}) & = & (\xi, a) \\ t' & (x_0, y_0, x_1, y_1, \ldots, x_{\ell-1}, y_{\ell-1}, z) & = & (\xi, b) \quad \text{for } z = 0, 1. \end{array}$$

It is straightforward to verify that $\nabla_i = \eta^{-1}[V_i]$, for $i = 1, 2$. Suppose that V_1, V_2 are separated by a set C of class Δ_m. It is easy to check that the preimage under the mapping η of a tree language recognized by an alternating automaton of an index (i, n) can be itself recognized by an automaton of the same index. Hence $\eta^{-1}[C]$ is in the class Δ_m and separates ∇_1 and ∇_2, which contradicts Theorem 6. ◀

References

1. Arnold, A., The μ-calculus alternation-depth hierarchy is strict on binary trees. *RAIRO-Theoretical Informatics and Applications* 33 (1999), 329–339.
2. Arnold, A., and Niwiński, D., Fixed point characterization of weak monadic logic definable sets of trees. In M.Nivat, A.Podelski, editors, *Tree Automata and Languages*, Elsevier, 1992, 159-188.
3. Arnold, A., and Niwiński, D., *Rudiments of μ-Calculus.* Elsevier Science, Studies in Logic and the Foundations of Mathematics, 146, North–Holland, Amsterdam, 2001.
4. Bradfield, J.C., Simplifying the modal mu-calculus alternation hierarchy. In: *Proc. STACS'98*, Lect. Notes Comput. Sci. 1373 (1998), 39–49.
5. Hummel, S., Michalewski, H., and Niwiński, D., On the Borel Inseparability of Game Tree Languages. STACS 2009:565-575.
6. Kaminski, M., A Classification of omega-Regular Languages. *Theor. Comput. Sci.* 36: 217-229 (1985).
7. Kechris, A.S., *Classical descriptive set theory.* Springer-Verlag, New York, 1995.
8. Moschovakis, Y. N., *Descriptive Set Theory.* North Holland, 1980.
9. Muller, D.E., Saoudi, A., and Schupp, P.E., Alternating Automata, the Weak Monadic Theory of Trees and its Complexity. *Theoret. Comput. Sci.* 97(2): 233-244 (1992).
10. Rabin, M.O., Weakly definable relations and special automata. In: *Mathematical Logic and Foundations of Set Theory*, Y. Bar-Hillel ed., 1970, 1-23.
11. Santocanale, L., and Arnold, A., Ambiguous classes in μ-calculi hierarchies. *Theoret. Comput. Sci.* 333 (2005), 265-296.
12. Selivanov, V., Fine hierarchy of regular ω-languages. *Theoret. Comput. Sci.* 191 (1998), 37–59.
13. Selivanov, V., Fine hierarchy and m-reducibilities theoretical computer science. *Theoret. Comput. Sci.* 405 (2008), 116–163.
14. Thomas, W., Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, Springer-Verlag, 1997, pp. 389–455.
15. Wagner, K., On ω-regular sets. *Information and Control*, 43:123–177 (1979).

On the treewidth and related parameters of random geometric graphs*

Dieter Mitsche[1] and Guillem Perarnau[2]

1 **Dept. of Mathematics, Ryerson University, 350 Victoria St., Toronto, ON, Canada.**
dmitsche@ryerson.ca

2 **MA4 - Universitat Politècnica de Catalunya, C/ Jordi Girona 1-3, 08034 Barcelona, Spain.**
guillem.perarnau@ma4.upc.edu

Abstract

We give asymptotically exact values for the treewidth $\mathrm{tw}(G)$ of a random geometric graph $\mathcal{G}(n,r)$ in $[0,\sqrt{n}]^2$. More precisely, we show that there exists some $c_1>0$, such that for any constant $0<r<c_1$, $\mathrm{tw}(G)=\Theta(\frac{\log n}{\log\log n})$, and also, there exists some $c_2>c_1$, such that for any $r=r(n)\geq c_2$, $\mathrm{tw}(G)=\Theta(r\sqrt{n})$. Our proofs show that for the corresponding values of r the same asymptotic bounds also hold for the pathwidth and treedepth of a random geometric graph.

1998 ACM Subject Classification G.2.2 Graph theory

Keywords and phrases Random geometric graphs, treewidth, treedepth

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.408

1 Introduction

Starting with the seminal paper of Gilbert [5], random geometric graphs have in recent decades received a lot of attention as a model for large communication networks such as sensor networks. Network agents are represented by the vertices of the graph, and direct connectivity is represented by edges. For applications of random geometric graphs, we refer to Chapter 3 of [7], and for a survey of many theoretical results, we refer to Penrose's monograph [12].

Given a set V of n vertices and a nonnegative real $r=r(n)$, a random geometric graph is defined as follows: each vertex is placed at some position of the square $S_n=[0,\sqrt{n}]^2$, chosen independently and uniformly at random. This choice of the square is only for convenience; by suitable scaling of r we could have chosen the square $[0,1]^2$ and the results were still valid.

Note that with probability 1 no two vertices choose the same position. We will identify each vertex with each position, that is, $u\in V$ refers also to the geometrical position of u in the square. Then we define $\mathcal{G}(n,r)$ as the random graph having V as the vertex set with $|V|=n$, and with an edge connecting each pair of vertices $u,v\in V$ at distance $d(u,v)\leq r$, where $d(\cdot,\cdot)$ denotes the Euclidean distance. In order to simplify calculations, we will use the well-known idea of Poissonization (see [12]): we assume that the vertices of $\mathcal{G}(n,r)$ are generated according to a Poisson point process of intensity 1 over the square $S_n=[0,\sqrt{n}]^2$.

* This research is partially supported by the Catalan Research Council under project 2009SGR1387. The first author is partially supported by the ICT Program of the European Union under contract number 215270 (FRONTS). The second author wants to thank the FPU grant from the Ministerio de Educación de España.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 408–419

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

Conditioned under the fact that this Poisson point process generates exactly n vertices (which happens with probability $\Theta(1/\sqrt{n})$), this model and the standard model of random geometric graphs have the same uniform distribution of the n vertices, and we will use this equivalence from now on.

All our stated results are asymptotic as $n \to \infty$. We use the usual notation *a.a.s.* to denote *asymptotically almost surely*, i.e. with probability $1 - o(1)$. It is well known that the property of the existence of a giant component of order $\Theta(n)$ undergoes a sharp threshold in $\mathcal{G}(n, r)$ (see e.g. [6]), but the exact value r is not yet known. However, there exist two positive constants c^-, c^+ such that for $r \leq c^-$, *a.a.s.* the largest component of $\mathcal{G}(n, r)$ is *a.a.s.* of order $O(\log n)$, whereas for $r \geq c^+$, a component of order $\Theta(n)$ is present (see [12]).

In this paper, we study the behaviour of two tree-like parameters, the treewidth and the treedepth, on random geometric graphs.

The treewidth of a graph measures the similarity between a tree and G. It was introduced by Robertson and Seymour in [17] inside their series of articles on graph minors. It has several applications in graph theory and algorithmics; one good example is Courcelle's Theorem [1].

For a graph $G = (V, E)$ on n vertices, we call (T, W) a *tree decomposition* of G, where W is a set of vertex subsets $W_1, \ldots, W_s \subseteq V(G)$ and T is a forest with vertices in W, such that

1. $\bigcup W_i = V(G)$
2. For any $e = uv \in E(G)$ there exists a set W_i such that $u, v \in W_i$
3. For any $v \in V(G)$, the subgraph induced by the $W_i \ni v$ is connected as a subgraph of T.

The *width* of a tree-decomposition is $\mathrm{w}(T, W) = \max_i |W_i| - 1$, and the *treewidth* of a graph G can be defined as

$$\mathrm{tw}(G) = \min_{(T,W)} \mathrm{w}(T, W).$$

A vertex partition $V = (A, S, B)$ is a *balanced k-partition* if $|S| = k + 1$, S separates A and B, and $\frac{1}{3}(n - k - 1) \leq |A|, |B| \leq \frac{2}{3}(n - k - 1)$. Then S is called a *balanced separator*. The following result connecting balanced partitions and treewidth is due to Kloks [9].

▶ **Lemma 1** ([9]). *Let G be a graph with n vertices and $\mathrm{tw}(G) \leq k$ such that $n \geq k - 4$. Then G has a balanced k-partition.*

The treedepth $\mathrm{td}(G)$ of a graph G was introduced by Nešetřil and Ossona de Mendez as a tree-like parameter in the scope of homomorphism theory. In particular, it provides an alternative definition of bounded expansion classes [11]. Moreover, the notion of the treedepth is closely connected to the treewidth. Intuitively speaking, the treewidth of a graph G is a parameter that measures the similarity between G and a certain tree, while the treedepth of G measures how close G is to a star. In other words, the treedepth also takes into account the diameter of the tree we are comparing the graph with.

This concept of treedepth has been introduced using different names in the literature. It is equivalent to the height of an elimination tree used in Cholesky decomposition [14]. Analogous definitions can be found using the terminology of rank function [10], vertex ranking number (or ordered coloring) [3] or weak coloring number [8].

Let T be a rooted tree. The *closure* of T is the graph that has the same set of vertices and two vertices are connected if they are relatives (ancestor or predecessor) in T. Consider a *rooted forest* as the disjoint union of rooted trees whose height is the maximum of the height among all the trees. The closure of a rooted forest will consist of the disjoint union of the closures of each rooted tree. The *treedepth* of a graph G, $\mathrm{td}(G)$, is defined to be the minimum height of a rooted forest, whose closure contains G as a subgraph.

Observe that, by definition, if G is a graph with components $H_1, \ldots, H_m$,

$$\mathrm{tw}(G) = \max_i \mathrm{tw}(H_i), \qquad \mathrm{td}(G) = \max_i \mathrm{td}(H_i). \tag{1}$$

This two parameters are closely related by the following inequalities:

$$\mathrm{tw}(G) \leq \mathrm{td}(G) \leq \mathrm{tw}(G)(\log n + 1),$$

both bounds being sharp. For example, if S is a star, $\mathrm{tw}(S) = \mathrm{td}(S) = 1$, while if P_n is a path of length n, $\mathrm{tw}(P_n) = 1$ and $\mathrm{td}(P_n) = \lfloor \log n \rfloor + 1$.

Results and organization of the paper. In this paper we study the values of $\mathrm{tw}(G)$ and $\mathrm{td}(G)$ of a random geometric graph $G = \mathcal{G}(n, r)$ for different values of $r = r(n)$. In particular, we prove the following two main theorems:

▶ **Theorem 2.** *There is some constant $0 < c_1 < c^-$, such that for any $0 < r \leq c_1$, a.a.s. $\mathrm{tw}(\mathcal{G}(n,r)) = \Theta(\frac{\log n}{\log\log n})$, and also a.a.s. $\mathrm{td}(\mathcal{G}(n,r)) = \Theta(\frac{\log n}{\log\log n})$.*

▶ **Theorem 3.** *There is some constant $c_2 > c^+$, such that for any $r = r(n) \geq c_2$, a.a.s. $\mathrm{tw}(G(n,r)) = \Theta(r\sqrt{n})$, and also a.a.s. $\mathrm{td}(\mathcal{G}(n,r)) = \Theta(r\sqrt{n})$.*

▶ Remark. For $G = \mathcal{G}(n, r)$ with r constant, but $r \geq c_2$, by the results of [2], many problems such as Steiner Tree, Feedback Vertex Set, Connected Vertex Cover can be solved in time $O(poly(n)3^{\sqrt{n}})$, and Connected Dominating Set, Connected Feedback Vertex Set, Min Cycle Cover, Longest Path, Longest Cycle, Graph Metric Travelling Salesman Problem can be solved in time $O(poly(n)4^{\sqrt{n}})$.

▶ Remark. Other width parameters that are sandwiched between treewidth and treedepth will have the same asymptotic behavior in $\mathcal{G}(n, r)$. For instance, the *pathwidth* of a graph, introduced by Robertson and Seymour [16], is defined to be the similarity between a graph and a path. Since the pathwidth is bounded from below by the treewidth and bounded from above by the treedepth (see Theorem 5.3 and Theorem 5.11 of [18]), the former theorems imply that for those values of $r = r(n)$ the pathwidth of the graph is of the same order.

We point out that it is an interesting feature of $\mathcal{G}(n, r)$ that treewidth and treedepth are asymptotically of the same order for a wide range of parameters r, since this is not true for random graphs in general [13]. The similar value of treedepth and treewidth implies that $\mathcal{G}(n, r)$ is more similar to a star–shaped tree than to a path–shaped tree, which in general is not true for random graphs. Observe also that in the period before the giant component the tree-like parameters are proportionally larger respect to the order of the components than when a giant component appears. In the classical random graph model the existence of a linear number of edges slightly above the giant component already implies a linear treewidth (see [4]), whereas a random geometric graph with the same number of edges (and a giant component) only has treewidth $\Theta(\sqrt{n})$.

In Section 2 we give the proof of Theorem 2. Whereas the lower bound follows from a standard argument about the maximum clique order, the proof of the upper bound is more involved. In Section 3 we continue by proving Theorem 3. Finally, in Section 4 we conclude mentioning open problems.

2 Proof of Theorem 2

Let $r_t = \Theta(1)$ the (not yet known) threshold radius of having a giant component, i.e. a connected component H with $V(H) = \Theta(n)$. In this section we will compute the treedepth

for a random geometric graph with $r < r_t$, i.e. when there is no giant component. We also assume $r = \Theta(1)$. From now on, unless otherwise stated, we will call the vertices of $\mathcal{G}(n,r)$ as *points*, since we use *vertex* for a different graph related to $\mathcal{G}(n,r)$ (see below). In [12] it is shown that the order of the largest component in this case is *a.a.s.* $\Theta(\log n)$, and we will assume this from now on. This implies directly the coarse upper bound $\mathrm{td}(G) = O(\log n)$.

For the sake of simplicity, we assume, moreover that $r < c_1$, where c_1 is a constant chosen in such a way that the order of each component is *a.a.s.* at most $\log n$ (this value exists, see Theorem 10.3 of [12], and is only chosen to simplify calculations).

We derive a lower bound on $\mathrm{tw}(G)$ by studying the clique number of G, $\omega(G)$. Tessellate S_n into square *cells* of side length $r/\sqrt{2}$. Note that we have a linear number of such cells and note that any two points in the same cell are connected by an edge. The distribution of the number of points inside the cells can be modeled as a *balls and bins* problem: we have n balls and $m = \Theta(n)$ bins, and each of the n balls is thrown independently and uniformly at random into one of the bins. Denoting X_i denote the number balls inside the cell C_i, classical results (see e.g. [15]) state that if $m = \Theta(n)$, then $\max_i X_i = (1 + o(1))\frac{\log n}{\log\log n}$ a.a.s..

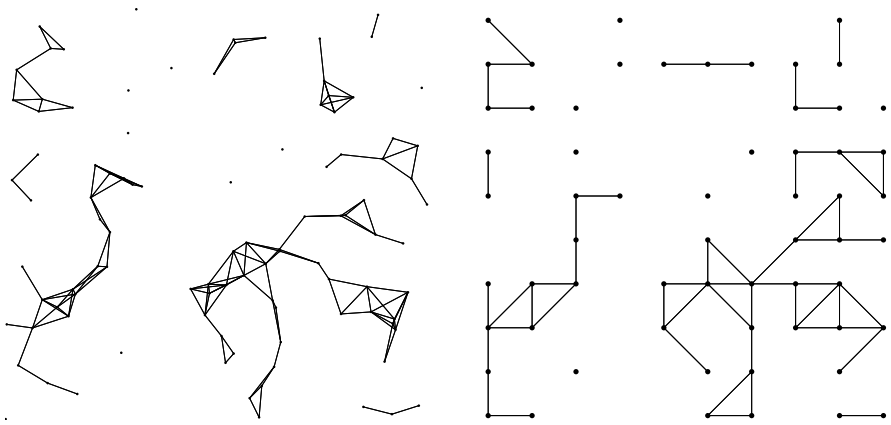
As any pair of points that belong to the same cell of the tessellation, is connected by an edge, G contains a clique subgraph formed of $\max_i X_i$ points, and therefore

$$\mathrm{td}(G) \geq \mathrm{tw}(G) \geq \omega(G) = \frac{\log n}{\log\log n}.$$

We will now show an upper bound on $\mathrm{td}(G)$ which asymptotically matches this lower bound. We use the following lemma.

▶ **Lemma 4.** *Let $X \sim \mathrm{Po}(\lambda)$. For any $k \geq 2\lambda$, $\Pr(X \geq k) \leq 2\Pr(X = k)$.*

Having tessellated S_n into cells, we construct a *cell-graph* C_G of G using the following criterion: each non-empty cell will be represented by a *vertex* and two vertices of C_G will be joined if there exist two points in the corresponding cells of G that share an edge (see Figure 1, where the tessellation is omitted for clarity). The cell-graph C_G has a structure similar to the original graph, but simpler.



(a) Random geometric graph **(b)** Cell-graph

Figure 1 A random geometric graph and its corresponding cell graph

Having in mind the previously established lower bound on the order of the maximum clique, set $\mathrm{T}_{\max} = \frac{\log n}{\log\log n}$. We focus on a certain connected component H of G that will

have order at most $\log n$. Note that there are at most n different components, not necessarily all of logarithmic order. Let C_H denote the cell-graph of component H. Note that since the side length is $\frac{r}{\sqrt{2}}$, each cell belongs to at most one connected component. Letting A_i be the number of points in the cell i (which will produce an A_i-clique) the number of points in H can be written as $\sum_{i\in V(C_H)} A_i$, and we have $\sum_{i\in V(C_H)} A_i \leq \log n$.

We will call a cell of the tessellation *sparse* if it contains less than $T = \frac{\sqrt{\log n}}{\log\log n}$ points, and *dense* otherwise. Observe that all the cells contain at most $\mathrm{T}_{\max}$ points.

▶ **Proposition 5.** *For any component H, the number of points belonging to dense cells is a.a.s. not larger than $O(\mathrm{T}_{\max})$.*

Proof. Since $A_i \sim \mathrm{Po}(\lambda)$, for some constant $\lambda = \lambda(r)$,

$$p = \Pr(A_i \geq T) \geq \Pr(A_i = T) \sim \frac{e^{-\lambda}}{\sqrt{2\pi T}} \left(\frac{e\lambda}{T}\right)^T, \tag{2}$$

using the Stirling approximation $T! \sim \sqrt{2\pi T}\left(\frac{T}{e}\right)^T$.

To count the number of points lying in dense cells, we define the following random variables:

$$Y_i = \begin{cases} t & \text{if } i \text{ is dense and has } t \text{ points inside} \\ 0 & \text{otherwise} \end{cases}$$

Our aim is to show that $Y = \sum_{i\in V(C_H)} Y_i$ is at most $O(\mathrm{T}_{\max})$. In this case (at least to us) it is not clear how a Chernoff type inequality can be used. Nevertheless, we will show that the probability that Y is larger than $8\,\mathrm{T}_{\max}$ is $o(n^{-1})$ and taking a union bound over all at most n components, *a.a.s.* no component will have more than $8\,\mathrm{T}_{\max}$ points in dense cells.

The probability of having a sparse cell is $1-p$, while the probability of having $T+j$ points inside a cell is $\Pr(\mathrm{Po}(\lambda) = T+j) \sim \frac{e^{-\lambda}}{\sqrt{2\pi(T+j)}}\left(\frac{e\lambda}{T+j}\right)^{T+j}$. Since $\frac{e^{-\lambda}}{\sqrt{2\pi(T+j)}}\left(\frac{e\lambda}{T+j}\right)^{T+j} \leq (\frac{e\lambda}{T})^T \frac{e^{-\lambda}}{\sqrt{2\pi T}}(\frac{e\lambda}{T})^j$, and using (2) we have

$$\Pr(\mathrm{Po}(\lambda) = T+j) \leq p\left(\tfrac{e\lambda}{T}\right)^j.$$

These observations lead to the definition of the following random variable:

$$R_i = \begin{cases} 0 & \text{with probability } 1-p. \\ T+j & \text{with probability } p\left(\frac{e\lambda}{T}\right)^j \text{ for any } j \geq 1. \\ T & \text{with probability } p\left(1 - \frac{e\lambda}{T-e\lambda}\right) \end{cases}$$

First of all, observe that R_i is a probability distribution. The random variables Y_i and R_i have similar distributions. In fact, each variable R_i stochastically dominates the corresponding random variable Y_i. Analogously we define $R = \sum_{i\in V(C_H)} R_i$. Then,

$$\Pr(Y > t) \leq \Pr(R > t) \text{ for any } t \in \mathbb{R} \tag{3}$$

and in particular this holds, if $t = O(\mathrm{T}_{\max})$.

Now we compute explicitly an upper bound for $\Pr(R > 8\,\mathrm{T}_{\max})$. We have $|V(C_H)| < \log n$ cells in H. There are n initial cells and then at most e^s different connected sets of s cells, and for this reason there are at most $ne^{\log n}$ ways to construct C_H. Assuming that i of them are dense, we have $\binom{|V(C_H)|}{i}$ ways to choose them, and after that, at most $(\log n)^i$

ways to distribute the points among these cells. The probability of having a dense cell with $R_i = T + j$ is $p\left(\frac{e\lambda}{T}\right)^j$, so that

$$\begin{aligned}\Pr(R > 8\,\mathrm{T}_{\max}) &= ne^{\log n}\sum_{i=1}^{|V(C_H)|}\sum_{S\in\binom{V(C_H)}{i}}\sum_{\sum_{j\in S}c_j\geq 8\,\mathrm{T}_{\max}}\Pr\left(\bigwedge_{j\in S}A_j = c_j\right)\\ &\leq n^2\sum_{i=0}^{|V(C_H)|}\binom{\log n}{i}(\log n)^i\prod_{j=1}^{i}p\left(\frac{e\lambda}{T}\right)^{c_j-T}.\end{aligned}$$

We use the upper bound $\binom{\log n}{i} \leq (\log n)^i$. It must be also stressed that we have

$$\prod_{j=1}^{i}p\left(\frac{e\lambda}{T}\right)^{c_j-T} \leq (p\sqrt{2\pi T})^{\frac{\sum c_j}{T}} < (p\sqrt{2\pi T})^{8\sqrt{\log n}},$$

since $\sum c_j > 8\,\mathrm{T}_{\max}$. Moreover, since $c_j > T$ (the cells are dense), we have for $i = 8\sqrt{\log n}+k$, $\prod_{j=1}^{i}p\left(\frac{e\lambda}{T}\right)^{c_j-T} \leq p^{8\sqrt{\log n}}p^k$. Therefore, it is useful to split the former equation into two sums:

$$\begin{aligned}\Pr(R > 8\,\mathrm{T}_{\max}) &\leq n^2\sum_{i=0}^{8\sqrt{\log n}}(\log n)^{2i}(p\sqrt{2\pi T})^{8\sqrt{\log n}}\\ &\quad +n^2\left((\log n)^2p\right)^{8\sqrt{\log n}}\sum_{k>0}\left((\log n)^2p\right)^k\end{aligned}$$

As $\left((\log n)^2p\right)^k < 1/2$, the infinite sum is less than one. Therefore,

$$\begin{aligned}\Pr(R > 8\,\mathrm{T}_{\max}) &\leq n^2\left(8\sqrt{\log n}+1\right)\left((\log n)^2\sqrt{2\pi T}p\right)^{8\sqrt{\log n}}\\ &\sim \exp\left\{2\log n + 4\log\log n + 8\sqrt{\log n}\,(2\log\log n + \log p + O(\log T))\right\}\end{aligned}$$

Since $p \sim \frac{c}{\sqrt{T}}\left(\frac{e\lambda}{T}\right)^T$, by Lemma 4 and (2), $\log p \sim -\frac{1}{2}\sqrt{\log n}$. The term $\Theta(\log T) = O(\log\log n)$ is negligible and thus,

$$\Pr(R > 8\,\mathrm{T}_{\max}) \leq \exp\{-(1+o(1))2\log n\} = O(n^{-2}). \tag{4}$$

By (3), this also implies that $\Pr(Y > 8\,\mathrm{T}_{\max}) = O(n^{-2})$, and by taking a union bound over all components, this implies that *a.a.s.* there is no component having more than $8\,\mathrm{T}_{\max}$ points inside dense cells. ◀

In order to obtain the desired matching upper bound, we need to construct a representation of the shape of the connected components which simplifies the structure. We now tessellate the square $[0, \sqrt{n}]^2$ into square cells of side length r. Proposition 5 also follows for this kind of tessellation since the size of the cells differs just by a constant factor. Consider now the cell graph C_G from this tessellation. Observe that the points belonging to a cell can only be connected by an edge to points in the same cell and to points in one of the at most 8 cells adjacent to that cell. Therefore, C_G will be a subgraph of the diagonal two-dimensional grid graph $L_{\sqrt{n},\sqrt{n}}$, where each cell is adjacent to the 8 cells surrounding it. The following proposition will be useful:

▶ **Proposition 6.** *Let $L_{m,n}$ be a diagonal two-dimensional grid graph and suppose that $m \leq n$. Then*

$$\mathrm{td}(L_{m,n}) \leq m \log n.$$

Fix a component C_H of C_G. We know that $|V(C_H)| \leq |V(H)| \leq \log n$ and hence, the diameter of C_H is at most $\log n$. Without loss of generality we may assume that each vertex is connected to all its 8 neighbouring cells provided that they are non empty. Take a vertex $v \in V(C_H)$ for which there exists some other vertex at distance the diameter of C_H. The vertices of C_H at distance d from v are said to be in the d-th floor. We also refer to the points inside the cells at distance d from v as points in the d-th floor.

We provide an elimination scheme for H. We want to find a balanced separator of this component (both parts will have linear order) that contains at most $\frac{\log n}{(\log\log n)^2}$ points. In particular, the separator set will be chosen among the different floors of C_H, corresponding to points that belong to cells at some fixed distance from v in C_H. Select the last floor f such that the number of points in lower floors is at most $|V(H)|/2$. Observe that this is always a separator that splits the graph H into two smaller pieces of order at most $|V(H)|/2$. If this separator of H has order at most $\frac{\log n}{(\log\log n)^2}$, we align in the elimination tree the points of the separator in a path, and we proceed recursively for the two subgraphs. The subtrees corresponding to these subgraphs are attached as children of the last node in the separator.

Suppose now that the floor f contains more than $\frac{\log n}{(\log\log n)^2}$ points of H. Then we can have many consecutive floors, before and after f, with more than $\frac{\log n}{(\log\log n)^2}$ points. However, since the order of the component H is at most $\log n$, there can be at most $(\log\log n)^2$ such floors.

Considering C_H, this implies that we have at most $(\log\log n)^2$ such consecutive floors containing more than $\frac{\log n}{(\log\log n)^2}$ points. Let us call the cell graph of these floors L'. Right after and before these floors we have two small cuts in C_H (meaning that they contain less than $\frac{\log n}{(\log\log n)^2}$ points), call them A' and B' respectively. We will recursively repeat this procedure for the two remaining parts A (the floors before A') and B (the floors after B') (see Fig.2). Observe that both A and B contain at most $|V(H)|/2$ points each (but they could contain much less, and in fact B could be empty).

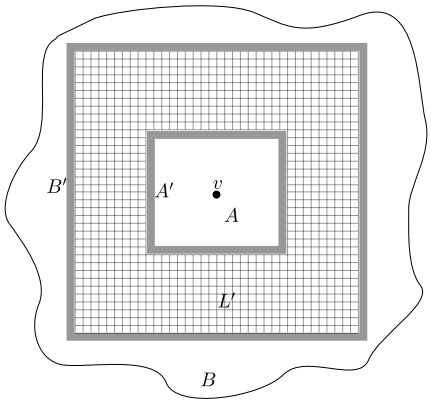


Figure 2 Decomposition of C_H

Focus now on L'. This is a subgraph of at most 4 copies of the diagonal grid $\log n \times (\log\log n)^2$ (see Fig.2), since there are at most $\log n$ points in each floor and therefore at

most $\log n$ cells containing them. By cutting these 4 copies and by using Proposition 6,

$$\mathrm{td}_{C_G}(L') \leq O\left((\log\log n)^3\right)$$

where td_{C_G} denotes the treedepth in the cell-graph.

The decomposition of $C_H = (A, A', L', B', B)$ gives the following inequality:

$$\mathrm{td}_{C_G}(C_H) \leq \frac{2\log n}{(\log\log n)^2} + \max\{\mathrm{td}_{C_G}(A), O\left((\log\log n)^3\right), \mathrm{td}_{C_G}(B)\}, \tag{5}$$

since, as A' and B' were two floors with few points inside, $|A' \cup B'| \leq \frac{2\log n}{(\log\log n)^2}$.

Observe that there exists $\alpha, \beta \leq 1/2$ such that $|A| \leq \alpha|V(H)|$ and $|B| \geq \beta|V(H)|$, and therefore, since the diameter of C_H is at most $\log n$, we can repeat this procedure at most $\log_2 |V(H)| = O(\log\log n)$ times. The constants α and β may change in each step but they are uniformly bounded by $1/2$. Hence, $\mathrm{td}_{C_G}(C_H) \leq O\left(\frac{\log n}{\log\log n}\right) = O(\mathrm{T}_{\max})$.

Now we are able to finish the proof of Theorem 2. By Proposition 5, we know that there are at most $O(\mathrm{T}_{\max})$ points in dense cells. We temporarily remove all these points, and add them at the end. Any of the remaining cells now has at most T points. We apply the previously described strategy of decomposition, the only difference being that each cell of L' contains now at most T points of G since there are no dense cells. Therefore, for the subgraph corresponding to L' in H we have $\mathrm{td}(L') \leq O\left(T(\log\log n)^3\right)$.

Since $T(\log\log n)^3 = o\left(\frac{2\log n}{(\log\log n)^2}\right)$, the upper bound on $\mathrm{td}(H)$ that arises from the formula (5) applied on the original graph, is not affected. Therefore, the treedepth of the component after removing the dense cells is at most $O(\mathrm{T}_{\max})$. Finally, taking into account all the points corresponding to the dense cells by attaching them all in a path above the root of the elimination tree for the non dense cells, we still have

$$\mathrm{td}(H) \leq O\left(\frac{\log n}{\log\log n}\right),$$

since adding a point increases the treedepth by at most 1. Using Equation (1), we have proven Theorem 2.

3 Proof of Theorem 3

Fix now $r = r(n) \geq c_2$, for some sufficiently large constant c_2 above r_t, the threshold radius of having a giant component. We will first give a strategy to construct an elimination tree for G, thus giving an upper bound on $\mathrm{td}(G)$.

Given $A \subseteq [0, \sqrt{n}]^2$, we denote by $\mathrm{vol}(A)$ the area of A. We need the following lemma:

▶ **Lemma 7.** *For any $A \subseteq [0, \sqrt{n}]^2$ such that $\mathrm{vol}(A) \geq c\log n$ and any $\delta > 0$, the number of points inside A is a.a.s. at most $(1+\delta)\,\mathrm{vol}(A)$.*

▶ **Proposition 8.** *For any $r \geq c_2$, $\mathrm{td}(G) \leq r\sqrt{n}$.*

Proof. We tessellate the square $[0, \sqrt{n}]^2$ into square cells of length r. Denote, moreover, by $C_{(i,j)}$ the j-th such cell in the i-th row, for $1 \leq i, j \leq m = \sqrt{n}/r$.

We provide some tree decomposition, such that G can be embedded as a subgraph of the closure of the tree. Define $X_1 = \cup_{i=0}^{m} C_{(\lfloor m/2 \rfloor, i)}$ and denote by $Y_1 = \{y_1, \ldots, y_s\}$ the points inside the cells of X_1 (in arbitrary order). We start constructing the tree by putting the root into y_1 and by attaching the path $y_1 - \cdots - y_s$. Next, we define $X_2^1 = \cup_{i=0}^{\lfloor m/2 \rfloor - 1} C_{(i, \lfloor m/2 \rfloor)}$

and $X_2^2 = \cup_{i=\lfloor m/2 \rfloor+1}^{m} C_{(i,\lfloor m/2 \rfloor)}$, and let $X_2 = \cup_{i=1}^{2} X_2^i$. Define X_i and X_i^j in the same way. At the end of the path $y_1 - \cdots - y_s$, we attach now two disjoint paths constructed with the points of Y_2^1 and Y_2^2, respectively (again in arbitrary order). This process will then be iteratively repeated until all the points are added to the tree (see Figure 3). Every two steps the number of cells in X_i grows by a factor of 2. If k is the number of steps, the construction ends when $2^{k/2} = \sqrt{n}/r$, that is, when $k = \log n - 2\log r$.

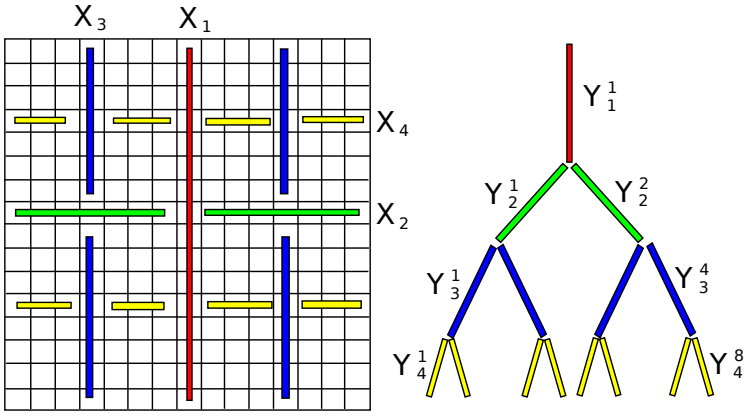


Figure 3 Sketch of the construction

Now we need to know the height of this elimination tree. Since $\mathrm{vol}(X_i)$ is at least of logarithmic size, by Lemma 7 we can always ensure the concentration on the number of points inside X_i^j.

Observe that in X_i^j there are $\frac{\sqrt{n}}{r} 2^{-\lceil (i+1)/2 \rceil}$ cells of the tessellation. Then, $\mathrm{vol}(X_i^j) = r^2 |X_i^j| = r\sqrt{n} 2^{-\lceil (i+1)/2 \rceil}$. For a sufficiently large c, if $i \leq \ell = \log n - 2\log\log n + 2\log r - \log c$, $\mathrm{vol}(X_i^j) \geq c \log n$ and by Lemma 7 together with a union bound over all j and $i \leq \ell$, we have *a.a.s.*

$$|Y_i^j| = O\left(r\sqrt{n} 2^{-\lceil (i+1)/2 \rceil}\right) \tag{6}$$

After this point $\mathrm{vol}(X_i)$ is too small to show concentration, but we have at most $k - \ell = 2\log\log n - 4\log r + \log c$ steps remaining. Since $\mathrm{vol}(X_i^j)$ beyond ℓ is smaller than $c \log n$, we will have at most the number of points inside an area of size $c \log n$ containing it. Thus, *a.a.s.*, for any j and $\ell \leq i \leq k$, $|Y_i^j| \leq O(\log n)$, and *a.a.s.*

$$\sum_{i=\ell}^{k} \max_j \left|Y_i^j\right| \leq O(\log n \log\log n).$$

Hence, the height of this elimination tree is *a.a.s.*

$$\begin{aligned} \mathrm{td}(G) &\leq \sum_{i=0}^{\ell} \max_j \left|Y_i^j\right| + \sum_{i=\ell+1}^{k} \max_j \left|Y_i^j\right| \\ &\leq O\left(r\sqrt{n}\left(\textstyle\sum_{i\geq 0} 2^{-\lceil (i+1)/2 \rceil}\right)\right) + O\left(\log n \log\log n\right) \\ &= O(r\sqrt{n}). \end{aligned}$$

◀

For convenience, tessellate the square $[0, \sqrt{n}]^2$ into small squares of size $r/4$. Given a set $A \subseteq V$ (identified with the corresponding geometric positions in $[0, \sqrt{n}]^2$), define by ∂A the boundary of A as $\partial A = \left\{x \in [0, \sqrt{n}]^2 : \min_{u \in A} d(x, u) = \frac{r}{2}\right\}$. We use $\mathrm{vol}(\partial A)$ to refer to the length of the boundary of A.

▶ **Lemma 9.** *Let S be a separator of the giant component. Let A be a connected component of $G \setminus S$. Then there exists a connected set of cells C_S containing $|C_S| = c_S = \Theta(\mathrm{vol}(\partial A)/r)$ cells, such that all the points inside C_S are from the giant component and in the separator.*

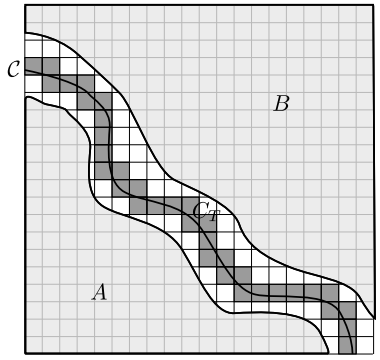


Figure 4 Cells of C_S

▶ **Theorem 10.** *There exists a constant c_2 such that for any $r \geq c_2$, a.a.s., $\mathrm{tw}(G) \geq \Omega(r\sqrt{n})$.*

Proof. We will show that there exists no balanced separator of size $o(r\sqrt{n})$ for the giant component H. Then, by Lemma 1, this implies that $\mathrm{tw}(H) = \Omega(r\sqrt{n})$, and therefore $\mathrm{tw}(G) \geq \mathrm{tw}(H) = \Omega(r\sqrt{n})$.

Let S be a fixed balanced separator of H. Let $S_1, \ldots, S_m$ the different connected components of S. If $m = \Omega(r\sqrt{n})$, for each component of S there is at least one point, since H is connected. This point belongs to S and to H and therefore the separator contains at least $m = \Omega(r\sqrt{n})$ points. Therefore we can assume that $m < r\sqrt{n}$.

Since S is balanced, there exist two sets A and B (not necessarily connected) with $|A| = \alpha n$, $|B| = \beta n$ for some $\frac{1}{3} < \alpha, \beta < \frac{2}{3}$ such that $G \setminus S$ contains no edges from A to B. By an isoperimetric inequality given a set A, $\mathrm{vol}(\partial A) = \Omega(\sqrt{\mathrm{vol}(A)})$. If $\mathrm{vol}(A) = \alpha n$ for $0 < \alpha < 1$, then even if A touches the boundary of $[0, \sqrt{n}]^2$, this is still true since at least a constant fraction of the perimeter is inside the square. Therefore we know that $\mathrm{vol}(\partial A) = \Omega(\sqrt{n})$, and by applying Lemma 9 for each connected component of S, we have a set of cells C_S with $c_S = \Omega(\sqrt{n}/r)$ such that all the points inside C_S are in S and in H.

Now we need to show that a.a.s. there are a lot of points inside C_S. Denote by Y the random variable counting the number of points inside C_S. The following simple claim shows that Y is concentrated around its expected value with very high probability.

▶ **Claim 11.** *The number of points Y inside C_S satisfies*

$$\Pr\left(Y < (1-\delta)\mathbf{E}(Y) = (1-\delta)\frac{r^2}{16}c_S\right) \leq e^{-\frac{\delta^2 r^2}{32}c_S}.$$

To show that no separator can have $o(r\sqrt{n})$ points we will use a union bound over all the possible balanced separators of H. Write $C_S = \cup C_{S_i}$ where C_{S_i} are the cells given by Lemma 9 for the separator S_i. Letting $c_{S_1}, \ldots, c_{S_m}$ the sizes of these separator components, there are at most $n^m e^{c_{S_1}+\cdots+c_{S_m}}$ ways to construct C_S: for each component C_{S_i} we have n places to choose where to start and then at most $e^{c_{S_i}}$ connected set of cells of size c_{S_i}.

Combining the previous upper bound from Claim 11 with a union bound over all separators of size $c_S \geq \Omega(\sqrt{n}/r)$, the probability of having such a bad balanced separator is at most

$$\sum_{c_S \geq \Omega(\sqrt{n}/r)} \sum_{m \leq O(r\sqrt{n})} \sum_{c_{S_1}+\cdots+c_{S_m}=c_S} n^m e^{c_S} e^{-\gamma r^2 c_S}, \tag{7}$$

where $\gamma = \delta^2/32$ for any $0 < \delta < \frac{1}{3}$. The number of ways to sum i using m non-negative numbers is $\binom{i+m}{m-1} \leq (i+m)^m \leq n^m$, and thus, (7) can be bounded from above by

$$\sum_{c_S \geq \Omega(\sqrt{n}/r)} \sum_{m \leq O(r\sqrt{n})} n^{2m} e^{c_S} e^{-\gamma r^2 c_S} \tag{8}$$

Observe that if $m \leq c\frac{r\sqrt{n}}{\log n}$ for some small constant $c > 0$, then $n^{2m} < e^{2cr\sqrt{n}} = o(e^{\gamma r^2 c_S})$, for sufficiently large γ. Therefore assume that $m > c\frac{r\sqrt{n}}{\log n}$.

Suppose that there is a constant fraction of cells in $C_G \setminus C_S$ contained in components of size at least $\frac{\sqrt{n}\log n}{cr}$. We restrict our separator to these big components. For this (sub)separator we have $m \leq c\left(\frac{\sqrt{n}}{r\log n}\right)$ (there are at most n/r^2 cells), and by the previous arguments, for this (sub)separator, the probability of having few points is at most $e^{-\gamma r^2 c_S}$ for some $\gamma > 0$, and hence the probability of having few points in S is also at most $e^{-\gamma r^2 c_S}$.

Thus, there is at least a constant fraction of vertices of $C_G \setminus C_S$ in components of order at most $\frac{\sqrt{n}\log n}{cr}$. Then, by the same isoperimetric inequality as before,

$$c_S \geq \frac{n^{1/4}\sqrt{\log n}}{\sqrt{cr}} \times c\frac{\sqrt{n}}{r\log n} = \Omega\left(\frac{n^{3/4}}{r^{3/2}\sqrt{\log n}}\right),$$

since all the components have order at least $\frac{\sqrt{n}\log n}{cr}$.

We distinguish two cases. First, we consider the case $c_2 \leq r = O(\sqrt{\log n})$. Since $m = O(r\sqrt{n})$, $n^{2m} = e^{2m\log n} \leq e^{2r\sqrt{n}\log n} \leq e^{2\sqrt{n}\log^{3/2} n}$ and $e^{\gamma r^2 c_S} \geq e^{\gamma \frac{n^{3/4}\sqrt{r}}{\sqrt{\log n}}} \geq e^{\gamma \frac{n^{3/4}}{\sqrt{\log n}}}$,

$$n^{2m} e^{c_S} e^{-\gamma r^2 c_S} \leq e^{-\gamma' r^2 c_S}$$

for some $0 < \gamma' < \gamma$. Otherwise, $r = \omega(\sqrt{\log n})$. Observe that $m \leq c_S$ since $c_{S_i} \geq 1$ by definition. Therefore,

$$n^{2m} e^{c_S} e^{-\gamma r^2 c_S} \leq n^{2c_S} e^{c_S} e^{-\gamma r^2 c_S} = e^{(2\log n + O(1) - \gamma r^2)c_S} \leq e^{-\gamma'' r^2 c_S}$$

for some $0 < \gamma'' < \gamma$. We showed that each term of (8) can be bounded by an exponentially small term. Hence, there exist constants $\nu, \nu' > 0$, such that with probability at most

$$\sum_{c_S \geq \Omega(\sqrt{n}/r)} \sum_{m \leq O(r\sqrt{n})} n^{2m} e^{c_S} e^{-\nu r^2 c_S} \leq O\left(rn^{3/2} e^{-\nu' r\sqrt{n}}\right) = o(1)$$

there exists a separator S containing less than $(1-\delta)\frac{r^2}{16}c_S = \Omega(r\sqrt{n})$ points connected to the giant component, completing the proof. ◀

4 Conclusion

We have shown that for random geometric graphs with $0 < r \leq c_1$ and for $r \geq c_2$ the parameters of treewidth and treedepth are asymptotically of the same order. The immediate natural question that remains open is whether for all values of $r = \Theta(1)$, including the values of $c_1 \leq r \leq c_2$, this happens to be true. For either of the parameters it would be interesting to know whether there is a sharp threshold width of order $o(1)$, in the sense that there exists some critical value of the radius r_c such that the treewidth (treedepth, respectively) of a graph with radius of at most $r_c - o(1)$ is of order $\Theta(\frac{\log n}{\log\log n})$ with probability at least $1-\epsilon$, and the treewidth (treedepth, respectively) of a graph with radius at least $r_c + o(1)$ is of

order $\Theta(\sqrt{n})$ with probability at least $1-\epsilon$, for any $\epsilon > 0$. We remark that the general result on sharp thresholds of monotone properties of [6] implies only a sharp threshold width of order $\log^{3/4} n$. Needless to say, in case of the existence of such a sharp threshold, it would be nice to find this exact threshold value for any of the two parameters (they might coincide). Using our methods, this, however, among other problems, requires the knowledge of the exact threshold value r_t of the appearance of the giant component in a random geometric graph, which at the moment is not known.

References

1 Bruno Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inform. and Comput.*, 85(1):12–75, 1990.

2 Marek Cygan, Jesper Nederlof, Marcin Pilipczuk, Michal Pilipczuk, Johan van Rooij, and Jakub Onufry Woitaszczyk. Solving connectivity problems parameterized by treewidth in single exponential time. *ArXiv e-prints*, March 2011.

3 Jitender S. Deogun, Ton Kloks, Dieter Kratsch, and Haiko Müller. On vertex ranking for permutation and other graphs. In *STACS 94 (Caen, 1994)*, volume 775 of *Lecture Notes in Comput. Sci.*, pages 747–758. Springer, Berlin, 1994.

4 Yong Gao. On the threshold of having a linear treewidth in random graphs. In *COCOON*, pages 226–234, 2006.

5 Edward N. Gilbert. Random plane networks. *Journal of the Society for Industrial and Applied Mathematics*, 9:533–543, 1961.

6 Ashish Goel, Sanatan Rai, and Bhaskar Krishnamachari. Sharp thresholds for monotone properties in random geometric graphs. In *STOC*, pages 580–586, 2004.

7 Ramin Hekmat. *Ad-hoc networks-fundamental properties and network topologies*. Springer, 2006.

8 Hal A. Kierstead and Daqing Yang. Orderings on graphs and game coloring number. *Order*, 20(3):255–264 (2004), 2003.

9 Ton Kloks. *Treewidth*, volume 842 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1994. Computations and approximations.

10 Jaroslav Nešetřil and Saharon Shelah. On the order of countable graphs. *European J. Combin.*, 24(6):649–663, 2003.

11 Jaroslav Nešetřil and Patrice Ossona de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European J. Combin.*, 27(6):1022–1041, 2006.

12 Mathew Penrose. *Random geometric graphs*, volume 5 of *Oxford Studies in Probability*. Oxford University Press, Oxford, 2003.

13 Guillem Perarnau and Oriol Serra. On the tree-depth of Random Graphs. *ArXiv e-prints*, April 2011.

14 Alex Pothen and Sivan Toledo. *Handbook on Data Structures and Applications*, chapter 59, Elimination structures in scientific computing. Chapman and Hall, 2004.

15 Martin Raab and Angelika Steger. "Balls into bins"—a simple and tight analysis. In *Randomization and approximation techniques in computer science (Barcelona, 1998)*, volume 1518 of *Lecture Notes in Comput. Sci.*, pages 159–170. Springer, Berlin, 1998.

16 Neil Robertson and Paul D. Seymour. Graph minors. I. Excluding a forest. *Journal of Combinatorial Theory*, 35(1):39–61, 1983.

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

18 Robert Sasak. Comparing 17 graph parameters. Master Thesis, Univ. of Bergen. 2010.

Optimizing Linear Functions with Randomized Search Heuristics – The Robustness of Mutation

Carsten Witt[1]

1 DTU Informatics, Technical University of Denmark

Abstract

The analysis of randomized search heuristics on classes of functions is fundamental for the understanding of the underlying stochastic process and the development of suitable proof techniques. Recently, remarkable progress has been made in bounding the expected optimization time of the simple (1+1) EA on the class of linear functions. We improve the best known bound in this setting from $(1.39 + o(1))en \ln n$ to $en \ln n + O(n)$ in expectation and with high probability, which is tight up to lower-order terms. Moreover, upper and lower bounds for arbitrary mutations probabilities p are derived, which imply expected polynomial optimization time as long as $p = O((\ln n)/n)$ and which are tight if $p = c/n$ for a constant c. As a consequence, the standard mutation probability $p = 1/n$ is optimal for all linear functions, and the (1+1) EA is found to be an optimal mutation-based algorithm. Furthermore, the algorithm turns out to be surprisingly robust since large neighborhood explored by the mutation operator does not disrupt the search.

1998 ACM Subject Classification F.2 [Analysis of algorithms and problem complexity]

Keywords and phrases Randomized Search Heuristics, Evolutionary Algorithms, Linear Functions, Running Time Analysis

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.420

1 Introduction

Consider the following modified Coupon Collector process. The n bins, initially all empty, have weights. At each time step, go through the bins and flip the state (full/empty) of each bin independently with probability $1/n$. Then check whether the total weight of the full bins has decreased compared to the previous time step. If so, restore the previous configuration, otherwise keep the new one. How long does it take until all bins are full at the same time?

If all bins weigh the same, then an $O(n \log n)$ bound on the expected time follows along the famous analysis of the Coupon Collector Problem. However, if the weights are different, then the analysis becomes much more involved. In fact, this problem has been studied for more than a decade in the analysis of randomized search heuristics (RSH) and is known as the linear function problem there.

RSHs are general problem solvers that may be used when no problem-specific algorithm is available. Famous examples are simulated annealing, evolutionary computation, tabu search etc. In order to understand the working principles of RSHs, and to give theoretically founded advice on the applicability of certain RSHs, rigorous analyses of the runtime of RSHs have been conducted. This is a growing research area where many results have been obtained in recent years. It started off in the early 1990's [15] with the consideration of very simple evolutionary algorithms such as the well-known (1+1) EA on very simple example functions such as the well-known OneMax function. Later on, results regarding the runtime on classes of functions were derived [9, 11, 19, 20, e. g.] and important tools for the analysis

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 420–431

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

were developed. Nowadays the state of the art in the field allows for the analysis of different types of search heuristics on problems from combinatorial optimization [16].

Recently, the analysis of evolutionary algorithms on linear functions has experienced a great renaissance. The first proof that the (1+1) EA optimizes any linear function in expected time $O(n \log n)$ by Droste et al. [9] was highly technical since it did not yet explicitly use the analytic framework of drift analysis [10], which allowed for a considerably simplified proof of the $O(n \log n)$ bound, see He and Yao [12] for the first complete proof using the method.[1] Another major improvement was made by Jägersküpper [13], who for the first time stated bounds on the implicit constant hidden in the $O(n \log n)$ term. This constant was finally improved by Doerr et al. [6] to the bound $(1.39 + o(1))en \ln n$ using a clean framework for the analysis of multiplicative drift [7]. The best known lower bound for general linear functions with non-zero weights is $en \ln n - O(n)$ and was also proven by Doerr et al. [6], building upon the OneMax function analyzed by Doerr et al. [3, 4].

The standard (1+1) EA flips each bit with probability $p = 1/n$ but also different values for the mutation probability p have been studied in the literature. Recently, it has been proved by Doerr and Goldberg [5] that the $O(n \log n)$ bound on the expected optimization time of the (1+1) EA still holds (also with high probability) if $p = c/n$ for an arbitrary constant c. This result uses the multiplicative drift framework mentioned above and a drift function being cleverly tailored towards the particular linear function. However, the analysis is also highly technical and does not yield explicit constants in the O-term. For $p = \omega(1/n)$, no runtime analyses were known so far.

In this paper, we prove that the (1+1) EA optimizes all linear functions in expected time $en \ln n + O(n)$, thereby closing the gap between the upper and the lower bound up to terms of lower order. Moreover, we show a general upper bound depending on the mutation probability p, which implies that the expected optimization time is polynomial as long as $p = O((\ln n)/n)$ (and $p = \Omega(1/\text{poly}(n))$). Since the expected optimization time is proved to be superpolynomial for $p = \omega((\ln n)/n)$, this implies a phase transition in the regime $\Theta((\ln n)/n)$. If the mutation probability is c/n for some constant c, the expected optimization time is proved to be $(1 \pm o(1))\frac{e^c}{c} n \ln n$. Altogether, we obtain that the standard choice $p = 1/n$ of the mutation probability is optimal for all linear functions. In fact, the lower bounds turn out to hold for the large class of so-called mutation-based EAs, in which the (1+1) EA with $p = 1/n$ is found to be an optimal algorithm.

Our findings are interesting both from a theoretical and practical perspective. On the theoretical side, it is noteworthy that $\frac{e^c}{c}$ is basically the expected waiting time for a mutation step that changes only a single bit. Hence, the mutation operator (in conjunction with the acceptance criterion) is surprisingly robust in the sense that steps flipping many bits do neither help nor harm. On the practical side, the optimality of $p = 1/n$ is remarkable since this seems to be the choice that is most often recommended by researchers in evolutionary computation [2]. Furthermore, the fact that the (1+1) EA is an optimal mutation-based algorithm emphasizes that it reflects the working principles of more complex EAs and that its runtime analysis can be crucial for obtaining results for more complex approaches.

The proofs of the upper bounds use the recent multiplicative drift theorem and a drift function adapted towards both the linear function and the mutation probability. As a consequence from our main result, we obtain the results by Doerr and Goldberg [5] with less effort and explicit constants in front of the $n \ln n$-term. All these bounds hold also with

[1] Note, however, that not the original (1+1) EA but a variant rejecting offspring of equal fitness is studied in that paper.

high probability, which follows from the recent tail bounds added to the multiplicative drift theorem by Doerr and Goldberg [5]. The lower bounds are based on a new multiplicative drift theorem for lower bounds. By deriving very exact results, we show that the research area is maturing and provides for very strong and, at the same time, general tools.

This paper is structured as follows. Section 2 sets up definitions, notations and other preliminaries. Section 3 summarizes and explains the main results. In Sections 4 and 5, respectively, we prove an upper bound for general mutation probabilities and a refined result for $p = 1/n$. Lower bounds are shown in Section 6. We finish with some conclusions. Due to space limitations, several proofs had to be omitted from this paper.

2 Preliminaries

The (1+1) EA is a basic search heuristic for the optimization of pseudo-boolean functions $f\colon \{0,1\}^n \to \mathbb{R}$. It reflects the typical behavior of more complicated evolutionary algorithms, serves as basis for the study of more complex approaches and is therefore intensively investigated in the theory of RSHs [1]. For the case of minimization, it is defined as Algorithm 1.

Algorithm 1 (1+1) EA

$t := 0$.
choose uniformly at random an initial bit string $x_0 \in \{0,1\}^n$.
repeat
  create x' by flipping each bit in x_t independently with prob. $p \leq 1/2$ *(mutation)*.
  $x_{t+1} := x'$ if $f(x') \leq f(x_t)$, and $x_{t+1} := x_t$ otherwise *(selection)*.
  $t := t + 1$.
until forever.

The (1+1) EA can be considered a simple hill-climber where search points are drawn from a stochastic neighborhood based on the mutation operator. The parameter p, where $0 < p \leq 1/2$, is often chosen as $1/n$, which then is called *standard mutation probability*. We call a mutation from x_t to x' *accepted* if $f(x') \leq f(x_t)$, i. e., if the new search point is taken over; otherwise we call it *rejected*. In our theoretical studies, we ignore the fact that the algorithm in practice will be stopped at some time. The *runtime* (synonymously, *optimization time*) of the (1+1) EA is defined as the first random point in time t such that the search point x_t has optimal, i. e., minimum f-value. This corresponds to the number of f-evaluations until reaching the optimum. In many cases, one is aiming for results on the expected optimization time. Here, we also prove results that hold with high probability, which means probability $1 - o(1)$.

The (1+1) EA is also an instantiation of the algorithmic scheme that is called *mutation-based EA* by Sudholt [17] and is displayed as Algorithm 2. It is a general population-based approach that includes many variants of evolutionary algorithms with parent and offspring populations as well as parallel evolutionary algorithms. Any mechanism for managing the populations, which are multisets, is allowed as long as the mutation operator is the only variation operator and follows the independent bit-flip property with probability $0 < p \leq 1/2$. Again the smallest t such that x_t is optimal defines the runtime. Sudholt has proved for $p = 1/n$ that no mutation-based EA can locate a unique optimum faster than the (1+1) EA can optimize OneMax. We will see that the (1+1) EA is the best mutation-based EA on a broad class of functions, also for different mutation probabilities.

Algorithm 2 Scheme of a mutation-based EA

for $t := 0 \to \mu - 1$ **do**
  choose $x_t \in \{0,1\}^n$ uniformly at random.
end for
repeat
  select a parent $x \in \{x_0, \dots, x_t\}$ according to t and $f(x_0), \dots, f(x_t)$.
  create x_{t+1} by flipping each bit in x independently with probability $p \leq 1/2$.
  $t := t + 1$.
until forever.

Throughout this paper, we deal with linear functions. A function $f\colon \{0,1\}^n \to \mathbb{R}$ is called *linear* if it can be written as $f(x_n, \dots, x_1) = w_n x_n + \cdots + w_1 x_1 + w_0$. As common in the analysis of the (1+1) EA, we assume w. l. o. g. that $w_0 = 0$ and $w_n \geq \cdots \geq w_1 > 0$ hold. Search points are read from x_n down to x_1 such that x_n, the most significant bit, is said to be on the left-hand side and x_1, the least significant bit, on the right-hand side. Since it fits the proof techniques more naturally, we assume also w. l. o. g. that the (1+1) EA (or, more generally, the mutation-based EA at hand) is minimizing f, implying that the all-zeros string is the optimum. Our assumptions do not lose generality since we can permute bits and negate the weights of a linear function without affecting the stochastic behavior of the (1+1) EA/mutation-based EA.

The probably best-studied linear function is $\textsc{OneMax}(x_n, \dots, x_1) = x_n + \cdots + x_1$, occasionally also called the *CountingOnes* problem (which would be the more appropriate name here since we will be minimizing the function). In this paper, we will see that on the one hand, OneMax is not only the easiest linear function definition-wise but also in terms of expected optimization time. On the other hand, the upper bounds obtained for OneMax hold for every linear function up to lower-order terms. Hence, surprisingly the (1+1) EA is basically as efficient on an arbitrary linear function as it is on OneMax. This underlines the robustness of the randomized search heuristic and, in retrospect and for the future, is a strong motivation to investigate the behavior of RSHs on the OneMax problem thoroughly.

Our proofs of the forthcoming upper bounds use the multiplicative drift theorem in its most recent version (cf. [5] and [7]). The key idea of multiplicative drift is to identify a time-independent relative progress called drift.

▶ **Theorem 1** (Multiplicative Drift, Upper Bound). *Let $S \subseteq \mathbb{R}$ be a finite set of positive numbers with minimum 1. Let $\{X^{(t)}\}_{t \geq 0}$ be a sequence of random variables over $S \cup \{0\}$. Let T be the random first point in time $t \geq 0$ for which $X^{(t)} = 0$.*

Suppose that there exists a $\delta > 0$ such that

$$E(X^{(t)} - X^{(t+1)} \mid X^{(t)} = s) \;\geq\; \delta s$$

for all $s \in S$ with $\text{Prob}(X^{(t)} = s) > 0$. Then for all $s_0 \in S$ with $\text{Prob}(X^{(0)} = s_0) > 0$,

$$E(T \mid X^{(0)} = s_0) \;\leq\; \frac{\ln(s_0) + 1}{\delta}.$$

Moreover, it holds that $\text{Prob}(T > (\ln(s_0) + t)/\delta)) \leq e^{-t}$.

As an easy example application, consider the (1+1) EA on OneMax and let $X^{(t)}$ denote the number of one-bits at time t. As worse search points are not accepted, $X^{(t)}$ is non-increasing over time. We obtain $E(X^{(t)} - X^{(t+1)} \mid X^{(t)} = s) \geq s(1/n)(1-1/n)^{n-1} \geq s/(en)$,

in other words a multiplicative drift of at least $\delta = 1/(en)$, since there are s disjoint single-bit flips that decrease the X-value by 1. Theorem 1 applied with $\delta = 1/(en)$ and $\ln(X^{(0)}) \leq \ln n$ gives us the upper bound $en(\ln n + 1)$ on the expected optimization time, which is basically the same as the classical method of fitness-based partitions [17, 18] or coupon collector arguments [14] would yield.

On a general linear function, it is not necessarily a good choice to let $X^{(t)}$ count the current number of one-bits. Consider, e. g., the well-known function $\textsc{BinVal}(x_n, \dots, x_1) = \sum_{i=1}^{n} 2^{i-1} x_i$. The (1+1) EA might replace the search point $(1, 0, \dots, 0)$ by the better search point $(0, 1, \dots, 1)$, amounting to a loss of $n-2$ zero-bits. More generally, replacing $(1, 0, \dots, 0)$ by a better search point is equivalent to flipping the leftmost one-bit. In such a step, an expected number of $(n-1)p$ zero-bits flip, which decreases the expected number of zero-bits by only $1-(n-1)p$. The latter expectation (the so-called additive drift) is only $1/n$ for the standard mutation probability $p = 1/n$ and might be negative for larger p. Therefore, $X^{(t)}$ is typically defined as $X^{(t)} := g(x^{(t)})$, where $x^{(t)}$ is the current search point at time t and $g(x_n, \dots, x_1)$ is another linear function called *drift function* or *potential function*. Doerr et al. [7] use $x_1 + \dots + x_{n/2} + (5/4)(x_{n/2+1} + \dots + x_n)$ as potential function in their application of the multiplicative drift theorem. This leads to a good lower bound on the multiplicative drift on the one hand and a small maximum value of $X^{(t)}$ on the other hand. In our proofs of upper bounds in the Sections 4 and 5, it is crucial to define appropriate potential functions.

For the lower bounds in Section 6, we need the following variant of the multiplicative drift theorem.

▶ **Theorem 2** (Multiplicative Drift, Lower Bound). *Let $S \subseteq \mathbb{R}$ be a finite set of positive numbers with minimum 1. Let $\{X^{(t)}\}_{t \geq 0}$ be a sequence of random variables over S, where $X^{(t+1)} \leq X^{(t)}$ for any $t \geq 0$, and let $s_{\min} > 0$. Let T be the random first point in time $t \geq 0$ for which $X^{(t)} \leq s_{\min}$. If there exist positive reals $\beta, \delta \leq 1$ such that for all $s > s_{\min}$ and all $t \geq 0$ with $\operatorname{Prob}(X^{(t)} = s) > 0$ it holds that*

1. $E\big(X^{(t)} - X^{(t+1)} \mid X^{(t)} = s\big) \leq \delta s$,
2. $\operatorname{Prob}(X^{(t)} - X^{(t+1)} \geq \beta s \mid X^{(t)} = s) \leq \beta\delta / \ln s$,

then for all $s_0 \in S$ with $\operatorname{Prob}(X^{(0)} = s_0) > 0$,

$$E\Big(T \mid X^{(0)} = s_0\Big) \geq \frac{\ln(s_0) - \ln(s_{\min})}{\delta} \cdot \frac{1-\beta}{1+\beta}.$$

The lower-bound version includes a condition on the maximum stepwise progress and requires monotonicity since the upper bound can be very pessimistic otherwise. As a technical detail, we allows for a positive target $s_{\min}$, which is required in our applications.

3 Summary of Main Results

We now list the main consequences from the lower bounds and upper bounds that we will prove in the following sections.

▶ **Theorem 3.** *On any linear function, the following holds for the expected optimization time $E(T_p)$ of the (1+1) EA with mutation probability p.*

1. *If $p = \omega((\ln n)/n)$ or $p = o(1/\operatorname{poly}(n))$ then $E(T_p)$ is superpolynomial.*
2. *If $p = \Omega(1/\operatorname{poly}(n))$ and $p = O((\ln n)/n)$ then $E(T_p)$ is polynomial.*
3. *If $p = c/n$ for a constant c then $E(T_p) = (1 \pm o(1))\frac{e^c}{c} n \ln n$.*
4. *$E(T_p)$ is minimized for mutation probability $p = 1/n$ if n is large enough.*
5. *No mutation-based EA has an expected optimization time that is smaller than $E(T_{1/n})$ (up to lower-order terms).*

In fact, our forthcoming analyses are more precise; in particular, we do not state available tails on the upper bounds above and leave them in the more general, but also more complicated Theorem 4 in Section 4. The first statement of our summarizing Theorem 3 follows from the Theorems 10 and 11 in Section 6. The second statement is proven in Corollary 6, which follows from the already mentioned Theorem 4. The third statement takes together the Corollaries 5 and 12. Since e^c/c is minimized for $c = 1$, the fourth statement follows from the third one in conjunction with Corollary 12. The fifth statement is also contained in the Theorems 10 and 11.

It is worth noting that the optimality of $p = 1/n$ apparently was never proven rigorously before, not even for the case of OneMax[2], where tight upper and lower bounds on the expected optimization time were only available for the standard mutation probability [4, 17]. For the general case of linear functions, the strongest previous result said that $p = \Theta(1/n)$ is optimal [9]. Our result on the optimality of the mutation probability $1/n$ is interesting since this is the commonly recommended choice by practitioners.

4 Upper Bounds

In this section, we show a general upper bound that applies to any non-trivial mutation probability.

▶ **Theorem 4.** *On any linear function, the optimization time of the (1+1) EA with mutation probability $0 < p \leq 1/2$ is at most*

$$(1-p)^{1-n}\left(\frac{n\alpha^2(1-p)^{1-n}}{\alpha-1} + \frac{\alpha}{\alpha-1}\frac{\ln(1/p)+(n-1)\ln(1-p)+t}{p}\right) =: b(t)$$

with probability at least $1-e^{-t}$, and it is at most $b(1)$ in expectation, where $\alpha > 1$ can be chosen arbitrarily (also depending on n).

Before we prove the theorem, we note two important consequences in more readable form. The first one (Corollary 5) displays upper bounds for mutation probabilities c/n. The second one (Corollary 6) is used in Theorem 3 above, which states a phase transition from polynomial to superpolynomial expected optimization times at mutation probability $p = \Theta((\ln n)/n)$.

▶ **Corollary 5.** *On any linear function, the optimization time of the (1+1) EA with mutation probability $p = c/n$, where $c > 0$ is a constant, is bounded from above by $(1+o(1))((e^c/c)n\ln n)$ with probability $1-o(1)$ and also in expectation.*

▶ **Corollary 6.** *On any linear function, the optimization time of the (1+1) EA with mutation probability $p = O((\ln n)/n)$ and $p = \Omega(1/\mathrm{poly}(n))$ is polynomial with probability $1-o(1)$ and also in expectation.*

The proof of Theorem 4 uses an adaptive potential function as in Doerr and Goldberg [5]. That is, the random variables $X^{(t)}$ used in Theorem 1 map the current search point of the (1+1) EA via a potential function to some value in a way that depends also on the linear function at hand. As a special case, if the given linear function happens to be OneMax, $X^{(t)}$ just counts the number of one-bits at time t. The general construction shares some

[2] However, a recent technical report extending Sudholt [17] shows the optimality of $p = 1/n$ in the case of OneMax using a different approach, see `http://arxiv.org/abs/1109.1504`.

similarities with the one in Doerr and Goldberg [5], but both construction and proof are significantly less involved. Moreover, we can also consider $p = \omega(1/n)$.

Proof of Theorem 4. Let $f(x) = w_n x_n + \cdots + w_1 x_1$ be the linear function at hand. Define

$$\gamma_i := \left(1 + \frac{\alpha p}{(1-p)^{n-1}}\right)^{i-1}$$

for $1 \le i \le n$, and let $g(x) = g_n x_n + \cdots + g_1 x_1$ be the potential function defined by $g_1 := 1 = \gamma_1$ and

$$g_i \;:=\; \min\left\{\gamma_i,\, g_{i-1} \cdot \frac{w_i}{w_{i-1}}\right\}$$

for $2 \le i \le n$. Note that the g_i are non-decreasing w. r. t. i. Intuitively, if the ratio of w_i and w_{i-1} is too extreme, the minimum function caps it appropriately, otherwise g_i and g_{i-1} are in the same ratio. We consider the stochastic process $X^{(t)} := g(a^{(t)})$, where $a^{(t)}$ is the current search point of the (1+1) EA at time t. Obviously, $X^{(t)} = 0$ if and only if f has been optimized.

Let $\Delta_t := X^{(t)} - X^{(t+1)}$. We will show below that

$$E(\Delta_t \mid X^{(t)} = s) \;\ge\; s \cdot p \cdot (1-p)^{n-1} \cdot \left(1 - \frac{1}{\alpha}\right). \qquad (*)$$

The initial value satisfies

$$X^{(0)} \;\le\; g_n + \cdots + g_1 \;\le\; \sum_{i=1}^{n} \gamma^i \;\le\; \frac{\left(1 + \frac{\alpha p}{(1-p)^{n-1}}\right)^n - 1}{\alpha p (1-p)^{1-n}} \;\le\; \frac{e^{n\alpha p (1-p)^{1-n}}}{\alpha p (1-p)^{1-n}},$$

which means

$$\ln(X^{(0)}) \;\le\; n\alpha p (1-p)^{1-n} + \ln(1/p) + \ln((1-p)^{n-1}).$$

The multiplicative drift theorem (Theorem 1) yields that the optimization time T is bounded from above by

$$\frac{\ln(X_0) + t}{p(1-p)^{n-1}(1-1/\alpha)} \;\le\; \frac{\alpha\left(n\alpha p(1-p)^{1-n} + \ln(1/p) + \ln((1-p)^{n-1}) + t\right)}{(\alpha-1)p(1-p)^{n-1}} \;=\; b(t)$$

with probability at least $1 - e^{-t}$, and $E(T) = b(1)$, which proves the theorem.

To show $(*)$, we fix an arbitrary current value s and an arbitrary search point $a^{(t)}$ satisfying $g(a^{(t)}) = s$. In the following, we implicitly assume $X^{(t)} = s$ but mostly omit this for the sake of readability. We denote by $I := \{i \mid a_i^{(t)} = 1\}$ the index set of the one-bits in $a^{(t)}$ and by $Z := \{1, \ldots, n\} \setminus I$ the zero-bits. We assume $I \neq \emptyset$ since there is nothing to show otherwise. Denote by a' the random (not necessarily accepted) offspring produced by the (1+1) EA when mutating $a^{(t)}$ and by $a^{(t+1)}$ the next search point after selection. Recall that $a^{(t+1)} = a'$ if and only if $f(a') \le f(a^{(t)})$. In the following, we will use the event A that $a^{(t+1)} = a' \neq a^{(t)}$ since obviously $\Delta_t = 0$ otherwise. Let $I^* := \{i \in I \mid a'_i = 0\}$ be the random set of flipped one-bits and $Z^* := \{i \in Z \mid a'_i = 1\}$ be the set of flipped zero-bits in a' (not conditioned on A). Note that $I^* \neq \emptyset$ if A occurs.

We need further definitions to analyze the drift carefully. For $i \in I$, we define $k(i) := \max\{j \le i \mid g_j = \gamma_j\}$ as the most significant position to the right of i (possibly i itself) where the potential function might be capping; note that $k(i) \ge 1$ since $g_1 = \gamma_1$. Let

$L(i) := \{k(i), \dots, n\} \cap Z$ be the set of zero-bits left of (and including) $k(i)$ and let $R(i) := \{1, \dots, k(i)-1\} \cap Z$ be the remaining zero-bits. Both sets may be empty. For event A to occur, it is necessary that there is some $i \in I$ such that bit i flips to zero and

$$\sum_{j \in I^*} w_j - \sum_{j \in Z^* \cap L(i)} w_j \geq 0$$

since we are taking only zero-bits out of consideration. Now, for $i \in I$, let A_i be the event that

1. i is the leftmost flipping one-bit (i. e., $i \in I^*$ and $\{i+1, \dots, n\} \cap I^* = \emptyset$) and
2. $\sum_{j \in I^*} w_j - \sum_{j \in Z^* \cap L(i)} w_j \geq 0$.

If none of the A_i occurs, $\Delta_t = 0$. Furthermore, the A_i are mutually disjoint.

For any $i \in I$, Δ_t can be written as the sum of the two terms

$$\Delta_L(i) := \sum_{j \in I^*} g_j - \sum_{j \in Z^* \cap L(i)} g_j \quad \text{and} \quad \Delta_R(i) := - \sum_{j \in Z^* \cap R(i)} g_j.$$

By the law of total probability and the linearity of expectation, we have

$$E(\Delta_t) = \sum_{i \in I} E(\Delta_L(i) \mid A_i) \cdot \text{Prob}(A_i) + E(\Delta_R(i) \mid A_i) \cdot \text{Prob}(A_i). \qquad (**)$$

In the following, the bits in $R(i)$ are pessimistically assumed to flip to 1 independently with probability p each if A_i happens. This leads to $E(\Delta_R(i) \mid A_i) \geq -p \sum_{j \in R(i)} g_j$.

In order to estimate $E(\Delta_L(i))$, we carefully inspect the relation between the weights of the original function and the potential function. By definition, we obtain $g_j/g_{k(i)} = w_j/w_{k(i)}$ for $k(i) \leq j \leq i$ and $g_j/g_{k(i)} \leq w_j/w_{k(i)}$ for $j > i$ whereas $g_j/g_{k(i)} \geq w_j/w_{k(i)}$ for $j < k(i)$. Hence, if A_i occurs then $g_j \geq g_{k(i)} \cdot \frac{w_j}{w_{k(i)}}$ for $j \in I^*$ (since i is the leftmost flipping one-bit) whereas $g_j \leq g_{k(i)} \cdot \frac{w_j}{w_{k(i)}}$ for $j \in L(i)$. Together, we obtain under $A(i)$ the nonnegativity of the random variable $\Delta_L(i)$:

$$\begin{aligned} \Delta_L(i) \mid A_i &= \sum_{j \in I^* \mid A_i} g_j - \sum_{j \in (Z^* \cap L(i)) \mid A_i} g_j \\ &\geq \sum_{j \in I^* \mid A_i} g_{k(i)} \cdot \frac{w_j}{w_{k(i)}} - \sum_{j \in (Z^* \cap L(i)) \mid A_i} g_{k(i)} \cdot \frac{w_j}{w_{k(i)}} \geq 0 \end{aligned}$$

using the definition of A_i.

Now let $S_i := \{|Z^* \cap L(i)| = 0\}$ be the event that no zero-bit from $L(i)$ flips. Using the law of total probability, we obtain that

$$\begin{aligned} E(\Delta_L(i) \mid A_i) \cdot \text{Prob}(A_i) &= E(\Delta_L(i) \mid A_i \cap S_i) \cdot \text{Prob}(A_i \cap S_i) \\ &\quad + E(\Delta_L(i) \mid A_i \cap \overline{S_i}) \cdot \text{Prob}(A_i \cap \overline{S_i}). \end{aligned}$$

Since $\Delta_L(i) \mid A_i \geq 0$, the conditional expectations are non-negative. We bound the second term on the right-hand side by 0. In conjunction with $(**)$, we get

$$E(\Delta_t) \geq \sum_{i \in I} E(\Delta_L(i) \mid A_i \cap S_i) \cdot \text{Prob}(A_i \cap S_i) + E(\Delta_R(i) \mid A_i) \cdot \text{Prob}(A_i).$$

Obviously, $E(\Delta_L(i) \mid A_i \cap S_i) \geq g_i$. We estimate $\text{Prob}(A_i \cap S_i) \geq p(1-p)^{n-1}$ since it is sufficient to flip only bit i and $\text{Prob}(A_i) \leq p$ since it is necessary to flip this bit. Further

above, we have bounded $E(\Delta_R(i) \mid A_i)$. Taking everything together, we get

$$E(\Delta_t) \geq \sum_{i \in I} \left(p(1-p)^{n-1} g_i - p^2 \sum_{j \in R(i)} g_j \right)$$
$$\geq \sum_{i \in I} \left(p(1-p)^{n-1} \frac{g_i}{g_{k(i)}} \gamma_{k(i)} - p^2 \sum_{j=1}^{k(i)-1} \gamma_j \right).$$

The term for i equals

$$p(1-p)^{n-1} \frac{g_i}{g_{k(i)}} \left(1 + \frac{\alpha p}{(1-p)^{n-1}}\right)^{k(i)-1} - \frac{p^2 \cdot \left(\left(1 + \frac{\alpha p}{(1-p)^{n-1}}\right)^{k(i)-1} - 1 \right)}{\left(\frac{\alpha p}{(1-p)^{n-1}}\right)}$$
$$\geq \left(1 - \frac{1}{\alpha}\right) p(1-p)^{n-1} \frac{g_i}{g_{k(i)}} \left(1 + \frac{\alpha p}{(1-p)^{n-1}}\right)^{k(i)-1} = \left(1 - \frac{1}{\alpha}\right) p(1-p)^{n-1} g_i,$$

where the inequality uses $g_i \geq g_{k(i)}$. Hence,

$$E(\Delta_t) \geq \sum_{i \in I} \left(1 - \frac{1}{\alpha}\right) p(1-p)^{n-1} g_i = \left(1 - \frac{1}{\alpha}\right) p(1-p)^{n-1} g(a^{(t)}),$$

which proves $(*)$, and, therefore, the theorem. □

5 Refined Upper Bound for Mutation Probability $1/n$

In this section, we consider the standard mutation probability $p = 1/n$ and refine the result from Corollary 5. More precisely, we obtain that the lower order-terms are $O(n)$. The proof is shorter and uses a simpler potential function.

▶ **Theorem 7.** *On any linear function, the expected optimization time of the (1+1) EA with $p = 1/n$ is at most $en \ln n + 2en + O(1)$, and the probability that the optimization time exceeds $en \ln n + (1+t)en + O(1)$ is at most e^{-t}.*

6 Lower Bounds

In this section, we state lower bounds that prove the results from Theorem 4 to be tight up to lower-order terms for a wide range of mutation probabilities. Moreover, we show that the lower bounds hold for the very large class of mutation-based algorithms (Algorithm 2). Recall that a list of the most important consequences is given above in Theorem 3. For technical reasons, we split the proof of the lower bounds into two main cases, namely $p = O(n^{-2/3-\varepsilon})$ and $p = \Omega(n^{\varepsilon-1})$ for any constant $\varepsilon > 0$. The proofs go back to OneMax as a worst case, as outlined in the following subsection.

6.1 OneMax as Easiest Linear Function

Doerr et al. [6] show with respect to the (1+1) EA with standard mutation probability $1/n$ that OneMax is the "easiest" function from the class of functions with unique global optimum, which comprises the class of linear functions. More precisely, the expected optimization time on OneMax is proved to be smallest within the class.

We will generalize this result to $p \leq 1/2$ with moderate additional effort. In fact, we will relate the behavior of an arbitrary mutation-based EA on OneMax to the (1+1) EA_μ in a similar way to Sudholt [17, Section 7]. The latter algorithm, displayed as Algorithm 3, creates search points uniformly at random from time 0 to time $\mu - 1$ and then chooses a best one from these to be the current search point at time $\mu - 1$; afterwards it works as the standard (1+1) EA. Note that we obtain the standard (1+1) EA for $\mu = 1$. Moreover, we will only consider the case $\mu = \text{poly}(n)$ in order to bound the running time of the initialization. This makes sense since a unique optimum (such as the all-zeros string for OneMax) is with overwhelming probability not found even when drawing $2^{\sqrt{n}}$ random search points.

Algorithm 3 (1+1) EA_μ

for $t := 0 \to \mu - 1$ **do**
  choose $x_t \in \{0,1\}^n$ uniformly at random.
end for
$x_t := \arg\min\{f(x) \mid x \in \{x_0, \ldots, x_t\}\}$ (breaking ties uniformly).
repeat
  create x' by flipping each bit in x_t independently with prob. p.
  $x_{t+1} := x'$ if $f(x') \leq f(x_t)$, and $x_{t+1} := x_t$ otherwise.
  $t := t + 1$.
until forever.

Our analyses need the monotonicity statement from Lemma 8 below, which is similar to Lemma 11 in Doerr et al. [6] and whose proof is already sketched in Droste et al. [8, Section 5]. Note, however, that Doerr et al. [6] only consider $p = 1/n$ and have a stronger statement for this case. More precisely, they show $\text{Prob}(|mut(a)|_1 = j) \geq \text{Prob}(|mut(b)|_1 = j)$, which does not hold for large p. Here and hereinafter, $|x|_1$ denotes the number of ones in a bit string x.

▶ **Lemma 8.** *Let $a, b \in \{0,1\}^n$ be two search points satisfying $|a|_1 < |b|_1$. Denote by $mut(x)$ the random string obtained by mutating each bit of x independently with probability p. Let $0 \leq j \leq n$ be arbitrary. If $p \leq 1/2$ then $\text{Prob}(|mut(a)|_1 \leq j) \geq \text{Prob}(|mut(b)|_1 \leq j)$.*

The following theorem is a generalization of Theorem 9 by Doerr et al. [6] to the case $p \leq 1/2$ instead of $p = 1/n$. However, we not only generalize to higher mutation probabilities, but also also consider the more general class of mutation-based algorithms. Finally, we prove stochastic ordering, while Doerr et al. [6] inspect only the expected optimization times. Still, many ideas of the original proof can be taken over and be combined with the proof of Theorem 5 in Sudholt [17].

▶ **Theorem 9.** *Consider a mutation-based EA A with population size μ and mutation probability $p \leq 1/2$ on any function with unique global optimum. Then the optimization time of A is stochastically at least as large as the optimization time of the (1+1) EA_μ on* OneMax.

6.2 Large Mutation Probabilities

It is not too difficult to show that mutation probabilities $p = \Omega(n^{\varepsilon-1})$, where $\varepsilon > 0$ is an arbitrary constant, make the (1+1) EA (and also the (1+1) EA_μ) flip too many bits for it to optimize linear functions efficiently.

▶ **Theorem 10.** *On any linear function, the optimization time of an arbitrary mutation-based EA with $\mu = \text{poly}(n)$ and $p = \Omega(n^{\varepsilon-1})$ for some constant $\varepsilon > 0$, is bounded from below by $2^{\Omega(n^\varepsilon)}$ with probability $1 - 2^{-\Omega(n^\varepsilon)}$.*

6.3 Small Mutation Probabilities

We now turn to mutation probabilities that are bounded from above by roughly $1/n^{2/3}$. Here quite precise lower bounds can be obtained.

▶ **Theorem 11.** *On any linear function, the expected optimization time of an arbitrary mutation-based EA with $\mu = \text{poly}(n)$ and $p = O(n^{-2/3-\varepsilon})$ is bounded from below by $(1 - o(1)) \cdot (1-p)^{-n}(1/p)\min\{\ln n, \ln(1/(p^3n^2))\}$.*

As a consequence from Theorem 11, we obtain that the bound from Theorem 4 is tight (up to lower-order terms) for the (1+1) EA as long as $\ln(1/(p^3n^2)) = \ln n - o(\ln n)$. This condition is weaker than $p = O((\ln n)/n)$. If $p = \omega((\ln n)/n)$ or $p = o(1/\text{poly}(n))$, then Theorem 11 in conjunction with Theorem 10 imply superpolynomial expected optimization time. Thus, the bounds are tight for all p that allow polynomial optimization times.

We state another important consequence, implying the statement from Theorem 3 that using the (1+1) EA with mutation probability $1/n$ is optimal for any linear function.

▶ **Corollary 12.** *On any linear function, the expected optimization time of a mutation-based EA with $\mu = \text{poly}(n)$ and $p = c/n$, where $c > 0$ is a constant, is bounded from below by $(1 - o(1))((e^c/c)n \ln n)$. If $p = \omega(1/n)$ or $p = o(1/n)$, the expected optimization time is $\omega(n \ln n)$.*

Finally, we remark that the expected optimization time of the (1+1) EA with $p = 1/n$ on OneMax is known to be $en \ln n - \Theta(n)$ [4]. Hence, in conjunction with the Theorems 7 and 9, we obtain for $p = 1/n$ that the expected optimization time of the (1+1) EA varies by at most an additive term $\Theta(n)$ within the class of linear functions.

Conclusions

We have presented new bounds on the expected optimization time of the simple (1+1) EA on the class of linear functions. The results are now tight up to lower-order terms, which applies to any mutation probability $p = O((\ln n)/n)$. This means that $1/n$ is the optimal mutation probability on any linear function. We have for the first time studied the case $p = \omega(1/n)$ and proved a phase transition from polynomial to exponential running time in the regime $\Theta((\ln n)/n)$. The lower bounds show that OneMax is the easiest linear function, and they apply not only to the (1+1) EA but also to the large class of mutation-based EAs. They so exhibit the (1+1) EA as optimal mutation-based algorithm on linear functions. The upper bounds hold with high probability. The analyses shed light on the working principles of randomized search heuristics on simple problems and prove that they can be surprisingly robust with respect to their parametrization. As proof techniques, we have used and further developed multiplicative drift analysis in conjunction with adaptive potential functions. In the future, we are confident to see these techniques applied to the analysis of other RSHs.

Acknowledgments

The author thanks Benjamin Doerr, Timo Kötzing, Per Kristian Lehre, Dirk Sudholt and Carola Winzen for insightful discussions and useful suggestions. Moreover, he thanks Daniel Johannsen for pointing out a simplification of the proof of Theorem 4.

References

1 Anne Auger and Benjamin Doerr. *Theory of Randomized Search Heuristics – Foundations and Recent Developments*. World Scientific Publishing, 2011.

2 Thomas Bäck. Optimal mutation rates in genetic search. In *Proc. of ICGA '93*, pages 2–8. Morgan Kaufmann, 1993.

3 Benjamin Doerr, Mahmoud Fouz, and Carsten Witt. Quasirandom evolutionary algorithms. In *Proc. of GECCO '10*, pages 1457–1464. ACM Press, 2010.

4 Benjamin Doerr, Mahmoud Fouz, and Carsten Witt. Sharp bounds by probability-generating functions and variable drift. In *Proc. of GECCO '11*, pages 2083–2090. ACM Press, 2011.

5 Benjamin Doerr and Leslie Ann Goldberg. Adaptive drift analysis. *Algorithmica*, 2012. To appear; preprint: `http://arxiv.org/abs/1108.0295`.

6 Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Drift analysis and linear functions revisited. In *Proc. of CEC '10*, pages 1–8. IEEE Press, 2010.

7 Benjamin Doerr, Daniel Johannsen, and Carola Winzen. Multiplicative drift analysis. In *Proc. of GECCO '10*, pages 1449–1456. ACM Press, 2010.

8 Stefan Droste, Thomas Jansen, and Ingo Wegener. A natural and simple funtions which is hard for all evolutionary algorithms. In *Proc. of IECON '00*, pages 2704–2709, 2000. DOI: 10.1109/IECON.2000.972425.

9 Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+1) evolutionary algorithm. *Theoretical Computer Science*, 276:51–81, 2002.

10 Bruce Hajek. Hitting-time and occupation-time bounds implied by drift analysis with applications. *Advances in Applied Probability*, 13(3):502–525, 1982.

11 Jun He and Xin Yao. Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence*, 127:57–85, 2001.

12 Jun He and Xin Yao. A study of drift analysis for estimating computation time of evolutionary algorithms. *Natural Computing*, 3(1):21–35, 2004.

13 Jens Jägersküpper. A blend of markov-chain and drift analysis. In *Proc. of PPSN '08*, volume 5199 of *LNCS*, pages 41–51. Springer, 2008.

14 Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

15 Heinz Mühlenbein. How genetic algorithms really work: I. Mutation and hillclimbing. In *Proc. of PPSN '92*, pages 15–26. Elsevier, 1992.

16 Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization – Algorithms and Their Computational Complexity*. Natural Computing Series. Springer, 2010.

17 Dirk Sudholt. General lower bounds for the running time of evolutionary algorithms. In *Proc. of PPSN '10*, volume 6238 of *LNCS*, pages 124–133. Springer, 2010. Extended version: `http://arxiv.org/abs/1109.1504`.

18 Ingo Wegener. Methods for the analysis of evolutionary algorithms on pseudo-Boolean functions. In Ruhul Sarker, Masoud Mohammadian, and Xin Yao, editors, *Evolutionary Optimization*. Kluwer Academic Publishers, 2001.

19 Ingo Wegener and Carsten Witt. On the analysis of a simple evolutionary algorithm on quadratic pseudo-boolean functions. *Journal of Discrete Algorithms*, 3(1):61–78, 2005.

20 Ingo Wegener and Carsten Witt. On the optimization of monotone polynomials by simple randomized search heuristics. *Combinatorics, Probability & Computing*, 14(1-2):225–247, 2005.

Parameterized Complexity of Connected Even/Odd Subgraph Problems*

Fedor V. Fomin[1] and Petr A. Golovach[2]

1 **Department of Informatics, University of Bergen**
PB 7803, 5020 Bergen, Norway
fedor.fomin@ii.uib.no

2 **School of Engineering and Computing Sciences, Durham University**
Science Laboratories, South Road, Durham DH1 3LE, UK
petr.golovach@durham.ac.uk

Abstract

Cai and Yang initiated the systematic parameterized complexity study of the following set of problems around Eulerian graphs. For a given graph G and integer k, the task is to decide if G contains a (connected) subgraph with k vertices (edges) with all vertices of even (odd) degrees. They succeed to establish the parameterized complexity of all cases except two, when we ask about

- a connected k-edge subgraph with all vertices of odd degrees, the problem known as k-Edge Connected Odd Subgraph; and
- a connected k- vertex induced subgraph with all vertices of even degrees, the problem known as k-Vertex Eulerian Subgraph.

We resolve both open problems and thus complete the characterization of even/odd subgraph problems from parameterized complexity perspective. We show that k-Edge Connected Odd Subgraph is FPT and that k-Vertex Eulerian Subgraph is W[1]-hard.

Our FPT algorithm is based on a novel combinatorial result on the treewidth of minimal connected odd graphs with even amount of edges.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems, G.2.1 Combinatorics, G.2.2 Graph Theory

Keywords and phrases Parameterized complexity, Euler graph, even graph, odd graph, treewidth

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.432

1 Introduction

An *even* graph (respectively, *odd* graph) is a graph where each vertex has an even (odd) degree. Recall that an Eulerian graph is a connected even graph. Let Π be one of the following four graph classes: Eulerian graphs, even graphs, odd graphs, and connected odd graphs. In [4], Cai and Yang initiated the study of parameterized complexity of subgraph problems motivated by Eulerian graphs. For each Π, they defined the following parameterized subgraph and induced subgraph problems:

* This work is supported by EPSRC (EP/G043434/1), Royal Society (JP100692), and the European Research Council (ERC) via grant Rigorous Theory of Preprocessing, reference 267959.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 432–440

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

k-Edge Π Subgraph (resp. k-Vertex Π Subgraph)	
Instance:	A graph G and non-negative integer k.
Parameter:	k.
Question:	Does G contain a subgraph with k edges from Π (resp. an induced subgraph on k vertices from Π)?

Cai and Yang established the parameterized complexity of all variants of the problem except k-Edge Connected Odd Subgraph and k-Vertex Eulerian Subgraph, see Table 1. It was conjectured that k-Edge Connected Odd Subgraph is FPT and k-Vertex Eulerian Subgraph is W[1]-hard. We resolve these open problems and confirm both conjectures.

	Eulerian	Even	Odd	Connected Odd
k-Edge	FPT [4]	FPT [4]	FPT [4]	FPT Thm. 3
k-Vertex	W[1]-hard Thm. 4	FPT [4]	FPT [4]	FPT [4]

Table 1 Parameterized complexity of k-Edge Π Subgraph and k-Vertex Π Subgraph.

The remaining part of the paper is organized as follows. In Section 2, we provide definitions and give preliminary results. In Section 3, we show that k-Edge Connected Odd Subgraph is FPT. Our algorithmic result is based on an upper bound for the treewidth of a minimal connected odd graphs with an even number of edges. We show that the treewidth of such graphs is always at most 3. The proof of this combinatorial result, which we find interesting in its own, is non-trivial and is given in Section 4. The bound on the treewidth is tight—complete graph on four vertices K_4 is a minimal connected odd graph with an even number of edges and its treewidth is 3. In Section 5, we prove that k-Vertex Eulerian Subgraph is W[1]-hard and observe that the problem remains W[1]-hard if we ask about (not necessary induced) Eulerian subgraph on k vertices. We conclude the paper in Section 6 with some open problems.

2 Definitions and Preliminary Results

Graphs. We consider finite undirected graphs without loops or multiple edges. The vertex set of a graph G is denoted by $V(G)$ and its edge set by $E(G)$. A set $S \subseteq V(G)$ of pairwise adjacent vertices is called a *clique*. For a vertex v, we denote by $N_G(v)$ its *(open) neighborhood*, that is, the set of vertices which are adjacent to v. Distance between two vertices $u, v \in V(G)$ (i.e., the length of the shortest (u,v)-path in the graph) is denoted by $\mathrm{dist}_G(u,v)$. For a vertex v and a positive integer k, $N_G^{(k)}[v] = \{u \in V(G) \mid \mathrm{dist}_G(u,v) \leq k\}$. The *degree* of a vertex v is denoted by $d_G(v)$, and $\Delta(G)$ is the maximum degree of G. For a set of vertices $S \subseteq V(G)$, $G[S]$ denotes the subgraph of G induced by S, and by $G - S$ we denote the graph obtained form G by the removal of all the vertices of S, i.e. the subgraph of G induced by $V(G) \setminus S$.

Parameterized Complexity. Parameterized complexity is a two dimensional framework for studying the computational complexity of a problem. One dimension is the input size n and another one is a parameter k. It is said that a problem is *fixed parameter tractable* (or FPT), if it can be solved in time $f(k) \cdot n^{O(1)}$ for some function f. One of basic assumptions of the Parameterized Complexity theory is the conjecture that the complexity class W[1] $\neq$ FPT,

and it is unlikely that a W[1]-hard problem could be solved in FPT-time. We refer to the books of Downey and Fellows [6], Flum and Grohe [7], and Niedermeier [8] for detailed introductions to parameterized complexity.

Treewidth. A *tree decomposition* of a graph G is a pair (X,T) where T is a tree and $X=\{X_i \mid i \in V(T)\}$ is a collection of subsets (called *bags*) of $V(G)$ such that:
1. $\bigcup_{i\in V(T)} X_i = V(G)$,
2. for each edge $\{x,y\} \in E(G)$, $x,y \in X_i$ for some $i \in V(T)$, and
3. for each $x \in V(G)$ the set $\{i \mid x \in X_i\}$ induces a connected subtree of T.

The *width* of a tree decomposition $(\{X_i \mid i \in V(T)\},T)$ is $\max_{i\in V(T)}\{|X_i|-1\}$. The *treewidth* of a graph G (denoted as $\mathbf{tw}(G)$) is the minimum width over all tree decompositions of G.

Minimal odd graphs with even number of edges. We say that a graph G is *odd* if all vertices of G are of odd degree. Let r be a vertex of G. We assume that G is *rooted* in r. Let G be a connected odd graph with an even number of edges. We say that G is a *minimal* if G has no proper connected odd subgraphs with an even number of edges containing r.

The importance of minimal odd subgraphs with even numbers of edges is crucial for our algorithm because of the following combinatorial result.

▶ **Theorem 1.** *Let G be a minimal connected odd graph with an even number of edges with a root r. Then $\mathbf{tw}(G) \leq 3$.*

For non-rooted graphs, we also have the following corollary.

▶ **Corollary 2.** *For any minimal connected odd graph G with an even number of edges, $\mathbf{tw}(G) \leq 3$.*

Let us remark that the bound in Theorem 1 is tight—complete graph K_4 with a root vertex r is a minimal odd graph with even number of edges and of treewidth 3. The proof of Theorem 1 is given in Section 4. This proof is non-trivial and technical, and we find the combinatorial result of Theorem 1 to be interesting in its own. From algorithmic perspective, Theorem 1 is a cornerstone of our algorithm; combined with color coding technique of Alon, Yuster and Zwick in [1] it implies that k-Edge Connected Odd Subgraph is FPT. We give this algorithm in the next section.

3 Algorithm for k-Edge Connected Odd Subgraph

To give an algorithm for k-Edge Connected Odd Subgraph, in addition to Theorem 1, we also need the following result of Alon, Yuster and Zwick from [1] obtained by a powerful color-coding technique.

▶ Proposition 1 ([1]). Let H be a graph on k vertices with treewidth t. Let G be a n-vertex graph. A subgraph of G isomorphic to H, if one exists, can be found in $O(2^{O(k)} \cdot n^{t+1})$ expected time and in $O(2^{O(k)} \cdot n^{t+1} \cdot \log n)$ worst-case time.

We are ready to prove the main algorithmic result of this paper.

▶ **Theorem 3.** *k-Edge Connected Odd Subgraph can be solved in time $O(2^{O(k \log k)} \cdot n^4 \cdot \log n)$ for n-vertex graphs.*

Proof. Let (G,k) be an instance of the problem. We apply the following algorithm.

Step 1. If k is odd and $\Delta(G) \geq k$, then return Yes. Else if k is odd but $\Delta(G) < k$, then go to Step 3.

Step 2. If k is even and $\Delta(G) \geq k$, then we enumerate all odd connected graphs H with k edges of treewidth at most 3. For each odd graph H of treewidth at most 3 and with k edges, we use Proposition 1 to check whether G has a subgraph isomorphic to H. The algorithm returns YES if such a graph H exists. Otherwise, we construct a new graph G by removing from the old graph G all vertices of degree at least k.

Step 3. For each vertex v, check whether there is a connected odd subgraph H with k edges that contains v. To do it, we enumerate all connected subgraphs with $p = 0, \ldots, k$ edges that include v using the following observation. For every connected subgraph H of G with $p \geq 1$ edges such that $v \in V(H)$, there is a connected subgraph H' with $p-1$ edges such that $v \in V(H')$ and H' is a subgraph of H. Hence, given all connected subgraphs with $p-1$ edges, we can enumerate all subgraphs with p edges by a brute-force algorithm. The algorithm returns YES if a connected odd subgraph H with k edges exists for some vertex v, and it returns NO otherwise. [1]

In what follows we discuss the correctness of the algorithm and evaluate its running time.

If k is odd and $\Delta(G) \geq k$, then the star $K_{1,k}$ is a subgraph of G. Hence, G has a connected odd subgraph with k edges.

Let k be even and let $r \in V(G)$ be a vertex with $d_G(r) \geq k$. If G has a connected odd subgraph with k edges containing r, then G has a minimal connected odd subgraph H with even number of edges rooted in r. Let $\ell = |E(H)|$. Graph H contains at most ℓ vertices in $N_G(r)$. It follows that there are $k - \ell$ vertices $v_1, \ldots, v_{k-\ell} \in N_G(r) \setminus V(H)$. Denote by H' the subgraph of G with the vertex set $V(H) \cup \{v_1, \ldots, v_{k-\ell}\}$ and the edge set $E(H) \cup \{rv_1, \ldots, rv_{k-\ell}\}$. Since k and ℓ are even, we have that H' is an odd graph. By Theorem 1, $\mathbf{tw}(H) \leq 3$. Graph H' is obtained from H by adding some vertices of degree 1, and, therefore, $\mathbf{tw}(H') \leq 3$. This means that when G has a connected odd subgraph H with k edges containing r, then there is a connected odd subgraph H' with k edges containing r and of treewidth at most three. But then in Step 2, we find such a graph H' with k edges.

If no connected odd subgraph with k edges was found in Step 2, then if such a graph exist, it contains no vertex of degree (in G) at least k. Therefore all such vertices can be removed from G without changing the solution. Finally, in Step 3, trying all possible connected subgraphs with k edges in the obtained graph of maximum degree at most $k-1$, we can deduce if G contains an odd subgraph with k edges.

Concerning the running time of the algorithm. There are at most $\binom{k(k-1)/2}{k}$ non-isomorphic graphs with k edges, and we can find all connected odd graphs with k edges in time $2^{O(k \log k)}$ and to check in time $O(k)$ if the treewidth of each of the graphs is at most three by making use of Bodlaender's algorithm [3]. The running time of this part can be reduced to $2^{O(k)}$, see e.g. [2]. Then for each graph H of this type, to check whether H is a subgraph of G, takes time $O(2^{O(k)} \cdot n^4 \cdot \log n)$ by Proposition 1.

When we arrive at Step 3, we have that $\Delta(G) \leq k-1$. We show by induction that for any $p \geq 1$, there are at most $p!k^p$ connected subgraphs with p edges that contain a given vertex v. Clearly, the claim holds for $p = 1$. Let $p > 1$. Any connected subgraph of G with $p-1$ edges has at most p vertices. Since there are at most pk possibilities to add an edge to this subgraph to obtain a connected subgraph with p edges, the claim follows. Therefore, for each vertex v, we can enumerate all connected subgraphs H with k edges that include v in

[1] The idea of Step 3 is due to anonymous STACS referee. This allows us to improve the running time $O(2^{O(k^2 \log k)} \cdot n^4 \cdot \log n)$ of the algorithm from the original version.

time $O(k!k^k)$. Hence, Step 3 can be done in time $O(2^{O(k \log k)} \cdot n)$. We conclude that the total running time of the algorithm is $O(2^{O(k \log k)} \cdot n^4 \cdot \log n)$. ◀

4 Minimal connected odd graphs with even number of edges

In this section we give a high level description of the proof of Theorem 1, the main combinatorial result of this paper. The proof is inductive, and for the inductive step we identify specific structures in a minimal connected odd graph with an even number of edges.

To proceed with the inductive step, we need a stronger version of Theorem 1. Let G be a graph and let $x \in V(G)$. We say that a graph G' is obtained from G by *splitting x into x_1, x_2*, if G' is constructed as follows: for a partition X_1, X_2 of $N_G(x)$, we replace x by two vertices x_1, x_2, and join x_1, x_2 with the vertices of X_1, X_2 respectively. The following claim implies Theorem 1.

▶ Claim 1. Let G be a minimal connected odd graph with an even number of edges with a root r. Then $\mathbf{tw}(G) \leq 3$.

Moreover, if $d_G(r) = 1$ and z is the unique neighbor of r, then at least one of the following holds:

i) there is a tree decomposition (X, T) of G of width at most three such that for any bag $X_i \in X$ with $z \in X_i$, $|X_i| \leq 3$; or
ii) for any graph G' obtained from $G - r$ by splitting z into z_1, z_2, $\mathbf{tw}(G') \leq 3$ and there is a tree decomposition (X, T) of G' of width at most three such that there is a bag $X_i \in X$ containing both z_1 and z_2.

To describe the structures in the graph, we need a notion of a subgraph with terminals. Roughly speaking, a subgraph with terminals is connected to the remaining part of the graph only via terminals. More formally, let H be a subgraph of graph G, and let $s_1, \ldots, s_r \in V(H)$. We say that H is a *subgraph of G with terminals $s_1, \ldots, s_r$* if there is a subgraph F of G such that

- $G = F \cup H$;
- $V(F) \cap V(H) = \{s_1, \ldots, s_r\}$; and
- $E(F) \cap E(H) = \emptyset$.

Thus every edge of G having at least one endpoint in a non-terminal vertex of H, should be an edge of H. In particular, terminal vertices of H separate non-terminal vertices of H from other vertices of G. We also say that a subgraph H with a given set of terminals is *separating* if the graph obtained from G by the removal of all non-terminal vertices of H and all the edges of H (denoted $G - H$) is not connected.

The specific structures we are looking for in the inductive step are the subgraphs isomorphic to graphs with terminals from the set $\mathcal{H} = \{H_1, H_2, H_3, H_4, H_5, H_6\}$ shown in Fig. 1. We often say that $H_i \in \mathcal{H}$ is contained in graph G (or G has H_i) if G has a subgraph isomorphic to H_i with the terminals shown in Fig. 1. Notice that H_6 is a subgraph of H_4 and H_5, and we are looking for H_6 only if we cannot find H_4 or H_5.

The proof of Claim 1 is by induction on the number of edges. The basis case is a graph with 6 edges. Every connected odd graph with an even number of edges has at least 6 edges, and there are only two graphs with 6 edges that have these properties, these graphs are shown in Fig. 2. Trivially, Claim 1 holds for these graphs for any choice of the root. Then we assume that a minimal connected odd graph G with an even number of edges has at least 8 edges.

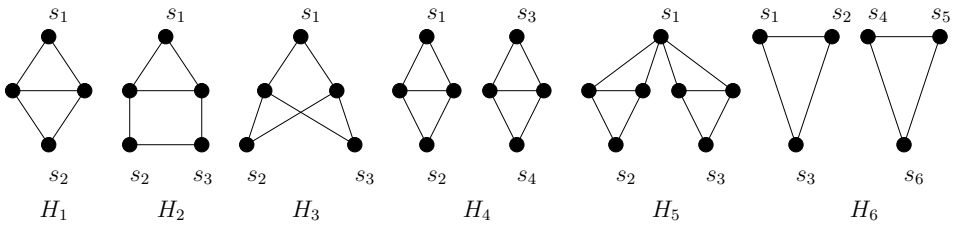


Figure 1 The set $\mathcal{H}$.

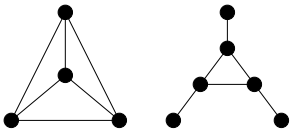


Figure 2 The base of the induction: Minimal graphs with six edges.

If G contains a subgraph R with terminals s_1, s_2 shown in Fig. 3 such that $r \notin V(R) \setminus \{s_1, s_2\}$ and $s_1s_2 \notin E(G)$, then we replace R by edge s_1s_2. It is possible to show that the resulting graph G' is a minimal connected odd graph with an even number of edges. Since G' has less edges than G, we can use the inductive assumption. Furthermore we assume that G has no R.

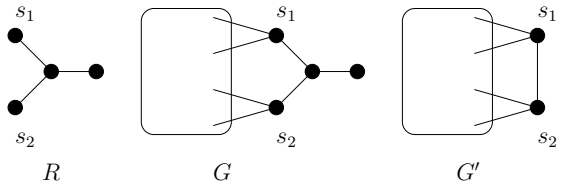


Figure 3 Replacement of R.

Next step is to prove that if G has no subgraph from $\mathcal{H}$, then G is one of the graphs G_1, G_2, G_3 shown in Fig. 4. For each of these graphs the theorem trivially holds. Actually, we will need a stronger result, saying that if G has no subgraph from $H_2, \ldots, H_6$ and every subgraph of G isomorphic to H_1 is of specific form, namely, this subgraph is not separating and r is not a non-terminal vertex of H_1, then even in this case, G is one of the graphs G_1, G_2, G_3 shown in Fig. 4. The proof of this claim is not straightforward. With this claim we can proceed further with an assumption that G contains at least one graph from $\mathcal{H}$.

For the case when r is a non-terminal vertex of a subgraph $H \in \mathcal{H}$, we prove that $H = H_1$. We remove non-terminal vertices of H, identify terminals s_1, s_2, and add a new root vertex r' adjacent to the vertex obtained from s_1, s_2. Then we prove that this graph is a minimal connected odd graph with an even number of edges, and then we can apply the induction assumption on this graph, and derive our claim for G. The difficulty here is to ensure that the treewidth of the graph G does not increase when we make the inductive step. This requires the assumptions i) and ii) in Claim 1 on the structure of tree decompositions. From this point, it can be assumed that r is not a non-terminal vertex of a subgraph from $\mathcal{H}$ with the corresponding set of terminals.

All graphs $H_2, \ldots, H_6$ have even number of edges and every terminal vertex of such a graph is of even degree. This means, that G cannot contain a non-separating graph H from $\{H_2, \ldots, H_6\}$, because removing edges and non-terminal vertices of H, would result in a

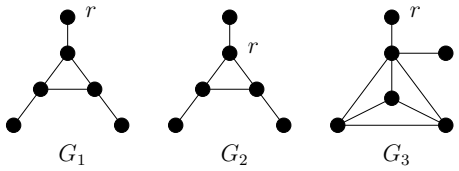


Figure 4 Graphs G_1, G_2, G_3.

connected odd subgraph of G with even number of edges, which is a contradiction to the minimality of G. Hence, if G contains subgraphs from $\mathcal{H}$ but they are non-separating, G can contain only H_1. Then as we already have shown, G is one of the graphs G_1, G_2, G_3 shown in Fig. 4.

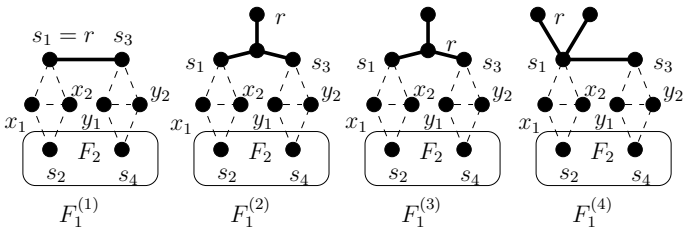


Figure 5 The case $H = H_4$, the trees $F_1^{(1)}, \ldots, F_1^{(4)}$ are formed by "bold" edges.

Thus we can assume that G contains a separating subgraph H from $\mathcal{H}$. Among all such separating subgraphs, we select H such that the number of edges of the component F_1 of the graph $G' = G - H$ containing r is minimum. We prove that G' has exactly two components F_1, F_2, where F_1 is a tree. We consequently consider the cases $H = H_1, \ldots, H_6$ and argue as follows. If $H = H_1$, then $F_1 = K_2$ and we apply induction for F_2 rooted in one of the terminals of H. If $H = H_2$, then we prove that $F_1 = K_2$. If $F_2 = K_2$, then the proof follows directly. Otherwise, we identify terminals s_1, s_3, and add a new root r' adjacent to the vertex obtained from s_1, s_3. It is possible to show that the constructed graph is a minimal connected odd graph with an even number of edges, and we can use the induction assumption for this graph. The arguments for the case $H = H_3$ are similar. If $H = H_4$, then we prove that F_1 is one of the trees $F_1^{(1)}, \ldots, F_1^{(4)}$ shown in Fig. 5. For F_2, we prove that $\mathbf{tw}(F_2) \leq 2$, and use this fact to construct a tree decomposition of G of width three. The case $H = H_5$ is similar. Finally, for $H = H_6$, we prove that it can be assumed that $s_1, s_2, s_4, s_5 \in V(F_1)$, $s_3, s_6 \in V(F_2)$, and F_1 is the tree shown in Fig. 6. Then we apply for F_2 the same arguments as in the case $H = H_4$. In each of the cases, we succeed to reduce G to a smaller minimal connected odd graph G' with even number of edges and show that $\mathbf{tw}(G) \leq \mathbf{tw}(G')$, which completes the induction step.

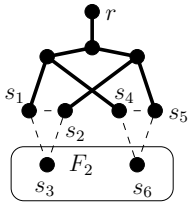


Figure 6 The case $H = H_6$, the tree F_1 is induced by "bold" edges.

5 Complexity of k-Vertex Eulerian Subgraph

In this section we prove that k-Vertex Eulerian Subgraph is W[1]-hard.

▶ **Theorem 4.** *The k-Vertex Eulerian Subgraph is W[1]-hard.*

Proof. We reduce from the well-known W[1]-complete k-Clique problem (see e.g. [6]):

k-Clique	
Instance:	A graph G and non-negative integer k.
Parameter:	k.
Question:	Does G contain a clique with k vertices?

Notice that the problem remains W[1]-complete when the parameter k is restricted to be odd. It follows immediately from the observation that the existence of a clique with k vertices in a graph G is equivalent to the existence of a clique with $k+1$ vertices in the graph obtained from G by the addition of a universal vertex adjacent to all the vertices of G. From now it is assumed that $k>1$ is an odd integer.

Let G be a graph. We construct the graph G' by subdividing edges of G by k^2 vertices, i.e. each edge xy is replaced by an (x,y)-path of length k^2+1. We say that $u\in V(G')$ is a *branch* vertex if $u\in V(G)$, and u is a *subdivision* vertex otherwise. We also say that u is a *subdivision vertex for an edge* $xy\in E(G)$ if u is a subdivision vertex of the path obtained from xy. We claim that G has a clique of size k if and only if G' has an induced Eulerian subgraph on $k'=\frac{1}{2}(k-1)k^3+k$ vertices.

Suppose that G has a clique K with k vertices. Let H be the subgraph of G induced by K and the subdivision vertices for all edges xy with $x,y\in K$. It is easy to see that H is a connected Eulerian graph on $k'=\frac{1}{2}(k-1)k^3+k$ vertices.

Let now H be an induced Eulerian subgraph of G' on $k'=\frac{1}{2}(k-1)k^3+k$ vertices. Denote by U the set of branch vertices of H, and let $p=|U|$. Let $A=\{xy\in E(G)|x,y\in U,$ and H has a subdivision vertex for $xy\}$ and let $F=(U,A)$. Let also $q=|A|$. Since H is connected, the graph F is connected as well. Observe that if $u\in V(H)$ is a subdivision vertex for an edge $xy\in E(G)$, then all subdivision vertices for xy are vertices of H and $x,y\in V(H)$. It follows that H has $p+q\cdot k^2=k'$ vertices, and we have $p-k=(\frac{1}{2}(k-1)k-q)k^2$. Since k^2 is a divisor of $p-k$, $p\geq k$. Suppose that $p>k$. Then since k^2 is a divisor of $p-k$, $p\geq k^2+k$. Any connected graph with p vertices has at least $p-1$ edges, and it means that $q\geq k^2+k-1>\frac{1}{2}(k-1)k$. We get that $0<p-k=(\frac{1}{2}(k-1)k-q)k^2<0$; a contradiction. We conclude that $p=k$. Then $q=\frac{1}{2}(k-1)k$ and U is a clique with k vertices. ◀

Recall that k-Vertex Eulerian Subgraph asks about an *induced* Eulerian subgraph on k vertices. For the graph G' in the proof of Theorem 4, any Eulerian subgraph is induced. It gives us the following corollary.

▶ **Corollary 5.** *The following problem:*

Instance:	*A graph G and non-negative integer k.*
Parameter:	*k.*
Question:	*Does G contain an Eulerian subgraph with k vertices?*

is W[1]*-hard.*

6 Conclusion

We proved that k-Edge Connected Odd Subgraph is FPT and k-Vertex Eulerian Subgraph is W[1]-hard. This completes the characterization of even/odd subgraph problems with *exactly* k edges or vertices from parameterized complexity perspective. While it is trivial to decide whether a graph G has a (connected) even or odd subgraph with *at most* k edges or vertices, the question about a subgraph with *at least* k edges or vertices seems to be much more complicated. For At Least k-Edge Odd Subgraph and At Least k-Vertex Odd Subgraph, following the lines of the proofs from [4] for k-Edge Odd Subgraph and k-Vertex Odd Subgraph, it is possible to show that these problems are in FPT. For other cases, the approaches used in [4] and in our paper, do not seem to work.

Cai and Yang in [4] also considered dual problems where the aim is to find an even or odd subgraph of a graph G with $|V(G)| - k$ vertices or $|E(G)| - k$ edges respectively. Recently, these results were complemented by Cygan et al. [5]. However, the complexity of the dual problem to k-Edge Connected Odd Subgraph, namely, obtaining connected odd subgraph with $|E(G)| - k$ edges, remains open.

Acknowledgments. The authors are grateful to the anonymous referees for their constructive suggestions and remarks.

References

1 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

2 Omid Amini, Fedor V. Fomin, and Saket Saurabh. Counting subgraphs via homomorphisms. In *Proceedings of the 36th International Colloquium on Automata, Languages and Programming (ICALP 2009)*, volume 5555 of *Lecture Notes Comp. Sci.*, pages 71–82. Springer, 2009.

3 Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

4 Leizhen Cai and Boting Yang. Parameterized complexity of even/odd subgraph problems. *J. Discrete Algorithms*, 9(3):231–240, 2011.

5 Marek Cygan, Dániel Marx, Marcin Pilipczuk, MichałPilipczuk, and Ildikó Schlotter. Parameterized complexity of eulerian deletion problems. In *Proceedings of the 37th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2011)*, volume 6986 of *Lecture Notes Comp. Sci.*, pages 131–142. Springer, 2011.

6 Rodney G. Downey and Michael R. Fellows. *Parameterized complexity.* Monographs in Computer Science. Springer-Verlag, New York, 1999.

7 Jörg Flum and Martin Grohe. *Parameterized complexity theory.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2006.

8 Rolf Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications.* Oxford University Press, Oxford, 2006.

Playing Mastermind With Constant-Size Memory

Benjamin Doerr[1] and Carola Winzen[2]

1 **Max-Planck-Institut für Informatik**
Saarbrücken, Germany
2 **Max-Planck-Institut für Informatik**
Saarbrücken, Germany

Abstract

We analyze the classic board game of Mastermind with n holes and a constant number of colors. The classic result of Chvátal (Combinatorica 3 (1983), 325–329) states that the codebreaker can find the secret code with $\Theta(n/\log n)$ questions. We show that this bound remains valid if the codebreaker may only store a constant number of guesses and answers. In addition to an intrinsic interest in this question, our result also disproves a conjecture of Droste, Jansen, and Wegener (Theory of Computing Systems 39 (2006), 525–544) on the memory-restricted black-box complexity of the OneMax function class.

1998 ACM Subject Classification F.2.2 [Analysis of algorithm and problem complexity]: Non-numerical Algorithms and Problems

Keywords and phrases Algorithms, Mastermind, black-box complexity, memory-restricted algorithms, query complexity

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.441

1 Introduction

The original *Mastermind* game is a board game for two players invented in the seventies by Meirowitz. It has pegs of six different colors. The goal of the *codebreaker*, for brevity called *Paul* here, is to find a color combination made up by *codemaker* (called *Carole* in the following). He does so by guessing color combinations and receiving information on how close this guess is to Carole's secret code. Paul's aim is to use as few guesses as possible.

For a more precise description, let us call the colors 1 to 6. Write $[n] := \{1, \dots, n\}$ for any $n \in \mathbb{N}$. Carole's secret code is a length-4 string of colors, that is, a $z \in [6]^4$. In each iteration, Paul guesses a string $x \in [6]^4$ and Carole replies with a pair $(\mathrm{eq}(z,x), \pi(z,x))$ of numbers. The first number, $\mathrm{eq}(z,x)$, which is usually indicated via black answer-pegs, is the number of positions, in which Paul's and Carole's string coincide. The other number, $\pi(z,x)$, usually indicated by white answer-pegs, is the number of additional pegs having the right color, but being in the wrong position. Formally $\mathrm{eq}(z,x) := |\{i \in [4] \mid z_i = x_i\}|$ and $\pi(z,x) := \max_{\rho \in S_4} |\{i \in [4] \mid z_i = x_{\rho(i)}\}| - \mathrm{eq}(z,x)$, where S_4 denotes the set of all permutations of [4]. Paul "wins" the game if he guesses Carole's string, that is, if Carole's answer is $(4,0)$.

We are interested in strategies for Paul that guarantee him to find the secret code with few questions. We thus adopt a worst-case view with respect to Carole's secret code. This is equivalent to assuming that Carole may change her hidden string at any time as long as it remains consistent with all previous answers (*devil's strategy*).

Previous results. Mathematics and computer science literature produces a plethora of results on the Mastermind problem. For the original game with 6 colors and 4 positions,

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 441–452
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Knuth [8] showed that Paul needs at most four queries until being able to identify Carole's string (which he may query in the fifth iteration to win the game).

Chvátal [3] studies a general version of this game with k colors and n positions, that is, the secret code is a length-n string $z \in [k]^n$. Denote by $d(n,k)$ the minimum number of guesses that enable Paul to win the game for any secret code. Chvátal proves that for $k < n^{1-\varepsilon}$, $\varepsilon > 0$ an arbitrarily small constant, we have $d(n,k) = O(\frac{n \log k}{\log n - \log k})$. More precisely, he shows that for any $\varepsilon > 0$ and n sufficiently large, $(2+\varepsilon)\frac{n(1+2\log k)}{\log n - \log k}$ guesses chosen from $[k]^n$ independently and uniformly at random, with high probability, suffice to distinguish between all possible codes (that is, each secret code leads to a different sequence of answers). Therefore, the secret code can be determined after that many guesses. This remains true if Carole replies only with black answer-pegs, that is, if for any of Paul's guesses x she reveals to him only $\mathrm{eq}(z,x)$, the number of bits, in which her and Paul's string coincide.

For larger values of k, the following is known. For $n \le k \le n^2$, Chvátal proves $d(n,k) \le 2n \log k + 4n$ and for $k = \omega(n^2 \log n)$ he shows $(k-1)/n \le d(n,k) \le \lceil k/n \rceil + d(n,n^2)$. These results have subsequently been improved. Chen, Cunha, and Homer [2] show that $d(n,k) \le 2n\lceil \log n \rceil + 2n + \lceil k/n \rceil + 2$ for $k \ge n$. Goodrich [7] proves $d(n,k) \le n\lceil \log k \rceil + \lceil (2-1/k)n \rceil + k$ for arbitrary k.

For $k = 2$ colors, the Mastermind problem is related to the well-studied coin weighing problem. For this reason, first results on this problem date back to years as early as 1963, when Erdős and Rényi [6] show that $d(n,2) = \Theta(n/\log n)$.

Concerning the computational complexity, Stuckman and Zhang [9] show that it is NP-hard to decide whether a given sequence $(x^{(i)}, (\mathrm{eq}^{(i)}, \pi^{(i)}))_{i=1}^t$ of queries $x^{(i)}$ and answers $(\mathrm{eq}^{(i)}, \pi^{(i)})$ of black and white pegs has a secret code leading to these answers, i.e., whether there exists a string $z \in [k]^n$ such that $\mathrm{eq}(z, x^{(i)}) = \mathrm{eq}^{(i)}$ and $\pi(z, x^{(i)}) = \pi^{(i)}$ for all $i \in [t]$. Goodrich [7] proves that this is already NP-hard if we only ask for consistence with the black answer-peg replies $\mathrm{eq}^{(i)}$.

Our results. Originally motivated by a conjecture on black-box complexities (cf. Section 2), we study a memory-restricted version of the Mastermind problem. Since this original motivation asks for the case of two colors only, we restrict ourselves to the number k of colors being constant, though clearly our methods can also be used to analyze larger numbers of colors.

The memory-restriction can be briefly described as follows. Given a memory of size $m \in \mathbb{N}$, Paul can store up to m guesses and Carole's corresponding replies. Based *only* on this information, Paul decides on his next guess. After receiving Carole's reply, based only on the content of the memory, the current guess, and the current answer, he decides which m out of the $m+1$ strings and answers he keeps in the memory. Note that our memory restriction means that Paul truly has no other memory, in particular, no iteration counters, no experience that certain colors are not used, and so one. So formally Paul's strategy consists of a guessing strategy, which can be fully described by a mapping from m-sets of guesses and answers to strings $x \in [k]^n$, and a forgetting strategy, which maps $(m+1)$-sets of guesses and answers to m-subsets thereof.

Clearly, a memory-restriction makes Paul's life not easier. The $O(n/\log n)$ strategies by Erdős and Rényi [6] and by Chvátal [3] do use the full history of guesses and answers and thus only work with a memory of size $\Theta(n/\log n)$. Surprisingly, this amount of memory is not necessary. In fact, one single memory cell is sufficient.

▶ **Theorem 1.** *Let $k \in \mathbb{N}$. For all $n \in \mathbb{N}$, Paul has a size-one memory strategy winning the Mastermind game with k colors and n positions in $O(n/\log n)$ guesses. This remains true if we allow Carole to play a devil's strategy and if Carole only reveals the number of fully correct pegs $\mathrm{eq}(x,z)$ ("black answer-peg version of Mastermind").*

The bound in Theorem 1 is asymptotically tight. A lower bound of $\Omega(n/\log n)$ is already true without memory restrictions. This follows easily from an information theoretic argument, cf. [6] or [3]. Our result disproves a conjecture of Droste, Jansen, and Wegener [5], who believed that a lower bound of $\Omega(n \log n)$ should hold for the 2-color black answer-peg Mastermind problem with memory-restriction one.

The proof of Theorem 1 is quite technical. For a clearer presentation of the ideas, we first consider the size-two memory-restricted model, cf. Section 3. The proof of Theorem 1 is given in Section 4. Before going into the proofs, in the following section we sketch the connection between Mastermind games and black-box complexities.

2 Mastermind and Black-Box Complexities

In this section, we describe the connection between the Mastermind game and black-box complexity. The reader only interested in the Mastermind result may skip this section without loss.

Roughly speaking, the *black-box complexity* of a set of functions is the number of function evaluations needed to find the optimum of an unknown member from that set. Since problem-unspecific search heuristics such as randomized hill-climbers, evolutionary algorithms, simulated annealing etc. do optimize by repeatedly generating new search points and evaluating their objective values ("fitness"), the black-box complexity is a lower bound on the efficiency of such general-purpose heuristics [5].

Black-box complexity. Let $\mathcal{S}$ be a finite set. A (randomized) algorithm following the scheme of Algorithm 1 is called black-box optimization algorithm for functions $\mathcal{S} \to \mathbb{R}$.

Algorithm 1: Scheme of a black-box algorithm for optimizing $f : \mathcal{S} \to \mathbb{R}$

```
Initialization: Sample x^(0) according to some probability distribution p^(0) on S;
Query f(x^(0));
for t = 1, 2, 3, ... do
    Depending on ((x^(0), f(x^(0))), ..., (x^(t-1), f(x^(t-1)))) choose a probability
    distribution p^(t) on S and sample x^(t) according to p^(t);
    Query f(x^(t));
```

For such an algorithm A and a function $f : \mathcal{S} \to \mathbb{R}$, let $T(A, f) \in \mathbb{R} \cup \{\infty\}$ be the expected number of fitness evaluations until A queries for the first time some $x \in \arg\max f$. We call $T(A, f)$ the *runtime of A for f*. For a class $\mathcal{F}$ of functions $\mathcal{S} \to \mathbb{R}$, the *$A$-black-box complexity of $\mathcal{F}$* is $T(A, \mathcal{F}) := \sup_{f \in \mathcal{F}} T(A, f)$, the worst-case runtime of A on $\mathcal{F}$. Let $\mathcal{A}$ be a class of black-box algorithms for functions $\mathcal{S} \to \mathbb{R}$. Then the *$\mathcal{A}$-black-box complexity of $\mathcal{F}$* is $T(\mathcal{A}, \mathcal{F}) := \inf_{A \in \mathcal{A}} T(A, \mathcal{F})$. If $\mathcal{A}$ is the class of all black-box algorithms, we also call $T(\mathcal{A}, \mathcal{F})$ the *unrestricted black-box complexity* of $\mathcal{F}$.

As said, the unrestricted black-box complexity is a lower bound for the efficiency of randomized search heuristics optimizing $\mathcal{F}$. Unfortunately, often this lower bound is not very useful. For example, Droste, Jansen, and Wegener [5] observe that the NP-complete MaxClique problem on graphs of n vertices has a black-box complexity of only $O(n^2)$.

Black-box algorithms with bounded memory. As a possible solution to this dilemma, Droste, Jansen, and Wegener suggest to restrict the class of algorithms considered from all black-box optimization algorithms to a reasonably large subset. A natural restriction is to forbid the algorithm to exploit the whole history of search points evaluated.

This is motivated by the fact that many heuristics, e.g., evolutionary algorithms, only store a bounded size *population* of search points. Simple hill-climbers or the Metropolis algorithm even store only one single search point.

Algorithm 2 is the scheme of a black-box algorithm with bounded memory of size μ. It is important to note that a black-box algorithm with bounded memory is not allowed to access any other information than the one stored in the μ pairs $(x^{(1)}, f(x^{(1)})), \dots, (x^{(\mu)}, f(x^{(\mu)}))$, which are currently stored in the memory and, in the selection step, also the information provided by $(x^{(\mu+1)}, f(x^{(\mu+1)}))$. In particular, the algorithm does not have access to an iteration counter.

Algorithm 2: Scheme of a black-box algorithm with memory of size μ for optimizing function $f : \mathcal{S} \to \mathbb{R}$

```
Initialization: M ← ∅;
for t = 1, 2, ... do
    Depending (only) on M choose a probability distribution p on S and sample x^(μ+1)
    according to p; //variation step
    Query f(x^(μ+1));
    Select M ⊆ M ∪ {(x^(μ+1), f(x^(μ+1)))} of size |M| ≤ μ; //selection step
```

Mastermind and the OneMax function class. A test function often regarded to analyze how the randomized search heuristic under investigation progresses in easy parts of the search space, is the simple OneMax function $\textsc{OneMax} : \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^n x_i$. Note that $\textsc{OneMax}(x) = \mathrm{eq}((1,\dots,1), x)$ for all $x \in \{0,1\}^n$. In fact, for any $z \in \{0,1\}^n$, $\mathrm{eq}(z, \cdot)$ yields an equivalent optimization problem. Let us denote by $\textsc{OneMax}_n := \{\mathrm{eq}(z, \cdot) \mid z \in \{0,1\}^n\}$ the class of all these functions.

Due to a coupon collector effect, many classical randomized search heuristics like randomized local search or the $(\mu + \lambda)$ evolutionary algorithm (with μ, λ constants) need $\Theta(n \log n)$ function evaluations to optimize $\textsc{OneMax}_n$.

As a moments thought reveals, black-box algorithms optimizing $\textsc{OneMax}_n$ correspond to strategies for Paul in the Mastermind game (without memory restriction) with two colors and only black answer-pegs used. Hence the unrestricted black-box complexity of $\textsc{OneMax}_n$ is $\Theta(n/\log n)$ by the results of Erdős and Rényi [6] and Chvátal [3].

This connection was seemingly overlooked so far in the randomized search heuristics community, where Droste, Jansen, and Wegener [5] prove an upper bound of $O(n)$ and later Anil and Wiegand [1] prove the asymptotically correct bound of $O(n/\log n)$. Since already the first bound is lower than what many randomized search heuristics achieve, Droste, Jansen, and Wegener suggest to investigate the memory-restricted black-box complexity of $\textsc{OneMax}_n$. They conjecture in [5, Section 6] that a memory restriction of size one leads to a black-box complexity of order $\Theta(n \log n)$.

Again, clearly, the memory-restricted black-box complexity of $\textsc{OneMax}_n$ and optimal strategies for Mastermind with two colors, black answer-pegs only, and a corresponding memory restriction are equivalent questions. Consequently, our result can be rephrased to saying that the black-box complexity of $\textsc{OneMax}_n$ even with the memory restricted to one is $\Theta(n/\log n)$, disproving the conjecture of Droste, Jansen, and Wegener.

3 The Mastermind Game with Memory of Size Two

Since the proof of Theorem 1 is quite technical, we give in this section a simpler proof showing that with a memory of size two Paul can win the game using only $O(n/\log n)$ guesses. Already this proof contains many ingredients needed to prove Theorem 1, e.g., the use of the random guessing strategy with limited memory, the block-wise determination of the secret code, and the simulation of iteration counters in the memory.

Let $k \geq 2$ be the number of colors used. In particular for $k = 2$, it will be convenient to label the colors from 0 to $k-1$. Let us denote the set of colors by $\mathcal{C} := [0..k-1] := \{0, 1, \dots, k-1\}$. We assume that k is a constant and that the number n of positions in the string is large, that is, all asymptotic notation is with respect to n.

▶ **Theorem 2.** *Paul has a size-two memory strategy winning the black answer-peg only Mastermind game with k colors and n positions in $O(n/\log n)$ guesses. This remains true if we allow Carole to play a devil's strategy.*

As many previous works, the proof of Theorem 2 heavily relies on *random guessing*. For the case of $k = 2$ colors, already Erdős and Rényi [6] showed that there is a $t \in \Theta(n/\log n)$ such that t guesses $x^{(1)}, \dots, x^{(t)}$ chosen from $\{0,1\}^n$ independently and uniformly at random, together with Carole's black answer-peg answers, uniquely define the hidden code. This was generalized by Chvátal [3] to the following result.

▶ **Theorem 3** (from [3]). *Let $\varepsilon > 0$, let $n > n(\varepsilon)$ be sufficiently large and let $k < n^{1-\varepsilon}$. Let $x^{(1)}, \dots, x^{(t)}$ be $t \geq (2+\varepsilon)\frac{n(1+2\log k)}{\log n - \log k}$ samples chosen from $\mathcal{C}^n$ independently and uniformly at random. Then for all $z \in \mathcal{C}^n$, the set*

$$\mathcal{S}^{consistent} := \{y \in \mathcal{C}^n \mid \forall i \in [t] : \mathrm{eq}(y, x^{(i)}) = \mathrm{eq}(z, x^{(i)})\}$$

satisfies $\mathrm{E}[|\mathcal{S}^{consistent}|] \leq 1 + 1/n$.

Since the strategy implicit in Theorem 3 needs a memory of size $\Theta(n/\log n)$, we cannot apply it directly in our setting. We can, however, adapt it to work on smaller portions ("blocks") of the secret code, and this with much less memory.

Let $y \in \mathcal{C}^n$ and let $B \subseteq [n]$ be a block (i.e., an interval) of size $s := \lceil \sqrt{n} \rceil$. As we shall see, by $t \in O(s/\log s)$ times guessing a string obtained from y by replacing the colors in B by randomly chosen ones (and guessing k additional *reference strings*), we can determine $z_{|B}$, the part of the secret code z in block B.

We can do so with a memory of size two only. We store the string obtained from y by altering it on B (*sampling string*) in one cell. Note that we do not need to remember y, as we only need to ensure that our guesses agree in the positions $[n] \setminus B$. We use the other memory cell (*storage string*, in the following typically denoted by x) to store the random substrings of length s substituted into y at B, and Carole's answers. Note that each such answer can be encoded in binary using $\ell_n \in O(\log n)$ entries of the string. Hence the t guesses and answers can be memorized using a total number of $t(s + \ell_n) = O(n/\log n)$ positions.

This approach allows us to determine s positions of z using $t = O(s/\log s)$ guesses. Hence we can determine the secret code z with $t\lceil n/s \rceil = O(n/\log n)$ guesses as desired.

In Algorithm 3 (notation used will be introduced below) we make this strategy more precise by giving it in pseudo-code. Note, however, that this algorithm does not fully satisfy the size-two memory restriction. The reason is that the queries do not only depend on the current state of the memory, but also on iteration counters and, e.g. in lines 9 and 11, on the program counter. Further below, in Algorithm 4 we shall remove this shortcoming with a few additional technicalities, which we are happy to spare for the moment.

Algorithm 3: An almost size-two memory-restricted algorithm winning the k-color black answer-peg only Mastermind game in $O(n/\log n)$ guesses. **Remark:** x denotes the unique string in $\mathcal{M}$ with $x_n = 1$ and y denotes the unique string in $\mathcal{M}$ with $y_n = 0$.

1 **Initialization:** $y \leftarrow [0 \dots 0]$;
2 Query $\mathrm{eq}(z, y)$ and update $\mathcal{M} \leftarrow \{(y, \mathrm{eq}(z, y))\}$;
3 **for** $i = 1$ **to** $\lceil (n-1)/s \rceil$ **do**
4 $x \leftarrow [0 \dots 0|1]$; //initialization of x
5 Query $\mathrm{eq}(z, x)$ and update $\mathcal{M}$ by adding $(i = 1)$ or replacing $(i > 1)$ $(x, \mathrm{eq}(z, x))$ in $\mathcal{M}$;
6 **for** $q = 0$ **to** $t + k - 1$ **do**
7 **if** $q < k$ **then** $y \leftarrow \mathtt{substitute}(y, B_i, [q \dots q])$; //reference string
8 **else** $y \leftarrow \mathtt{substitute}(y, B_i, r)$ where $r \in \mathcal{C}^{|B_i|}$ u.a.r.; //random guess
9 Query $\mathrm{eq}(z, y)$ and update $\mathcal{M}$ by replacing $(y, \mathrm{eq}(z, y))$;
10 $x \leftarrow [x_1 \dots x_{p_1(x)} | \mathtt{BLOCK}_i(y) | \mathtt{binary}_{\ell_n}(\mathrm{eq}(z, y)) | 1 | 0 \dots 0 | 1]$; //add y's info to x
11 Query $\mathrm{eq}(z, x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z, x))$;
12 **while** $\Delta_i(y) < |B_i|$ **do**
13 $y \leftarrow \mathtt{substitute}(y, B_i, w)$, where $w \in \mathcal{S}_i^{\text{consistent}}$ u.a.r.;
14 Query $\mathrm{eq}(z, y)$ and update $\mathcal{M}$ by replacing $(y, \mathrm{eq}(z, y))$;
15 **while** $\mathrm{eq}(z, y) < n$ **do** $y \leftarrow \mathtt{substitute}(y, \{n\}, c)$, where $c \in \mathcal{C}$ u.a.r., and query $\mathrm{eq}(z, y)$;

Before we argue for the correctness of Algorithm 3, let us fix the notation. For any string $x \in \mathcal{C}^n$ we also write $x = [x_1 \dots x_n]$. To ease reading, we allow ourselves to indicate different structural components of x by vertical bars, e.g., $x = [x_1 \dots x_p | x_{p+1} \dots x_n]$. For $i \in [\lceil (n-1)/s \rceil]$ let $B_i := \{(i-1)s + 1, \dots, is\} \cap [n-1]$, the positions of the i-th block. Set

$$\mathtt{BLOCK}_i(x) := x_{|B_i} := [x_{(i-1)s+1} \dots x_{\min\{is, n-1\}}] ,$$

the i-th block of x. For any string $r \in \mathcal{C}^{|B_i|}$ we define

$$\mathtt{substitute}(x, B_i, r) := [x_1 \dots x_{(i-1)s} | r | x_{\min\{is, n-1\}+1} \dots x_n] ,$$

the string with the i-th block substituted by r. Similarly, let $\mathtt{substitute}(y, \{n\}, c) := [y_1 \dots y_{n-1} | c]$. Note that we do not assign the n-th position to any of the blocks. We do so because in Algorithms 3 and 4 we shall use that position to indicate, which one of the two strings in the memory $\mathcal{M}$ is the storage string (the unique $x \in \mathcal{M}$ with $x_n = 1$) and which one is the sampling string (the unique string $y \in \mathcal{M}$ with $y_n = 0$).

Let $p_1(x) := \max\{i \in [n-1] \mid x_i = 1\}$, the largest position $i < n$ of x with entry "1". As mentioned above, we encode Carole's answers $\mathrm{eq}(z, y) \in [0..n]$ in binary, using $\ell_n := \lceil \log n \rceil + 1$ positions, and we denote this binary encoding of length ℓ_n by $\mathtt{binary}_{\ell_n}(\mathrm{eq}(z, y))$. By $\Delta_i(y)$ we denote the contribution of the i-th block to the value $\mathrm{eq}(z, y)$, i.e., $\Delta_i(y)$ is the number of positions in the i-th block, in which Paul's guess y and Carole's secret code z coincide. Formally, $\Delta_i(y) := \mathrm{eq}(z_{|B_i}, y_{|B_i})$. Lastly, let $\mathcal{S}_i^{\text{consistent}}$ be the set of strings w of length $|B_i|$ such that $\mathtt{substitute}(z, B_i, w)$ is consistent with all of Carole's replies (formal definition follows). We shall see below that both $\Delta_i(y)$ and $\mathcal{S}_i^{\text{consistent}}$ can be computed solely from the content of the memory cells (lines 12–14).

We now argue for the correctness of Algorithm 3. Let us consider the state of the memory after having sampled all t random samples for the i-th block (that is, we are in lines 12–14).

We show that based on the information given in the memory, we can restore the full history of guesses for the i-th block. To this end, first note that for any guess y done in line 9, we used $s + \ell_n + 1$ positions in x for storing its information (line 10; we add the additional "1" at the end to ease determining via $p_1(x)$ the positions in x, which have not yet been used for storing information). In lines 6–11 we first asked and stored k non-random guesses $x^c = \texttt{substitute}(y, B_i, [c \dots c])$ and we stored these *reference strings* together with Carole's replies $\mathrm{eq}(z, x^c) = \sum_{h=1}^{\ell_n} 2^{h-1} x_{c(s+\ell_n+1)-h}$, $c \in [0..k-1]$. Therefore, for $j \in [t]$, the j-th random sample is $r^{(j)} = [x_{(k+j-1)(s+\ell_n+1)+1} \dots x_{(k+j-1)(s+\ell_n+1)+|B_i|}]$ and the corresponding query was $y^{(j)} = \texttt{substitute}(y, B_i, r^{(j)})$. We have stored Carole's reply to this guess in binary, and we can infer $\mathrm{eq}(z, y^{(j)}) = \sum_{h=1}^{\ell_n} 2^{h-1} x_{(k+j)(s+\ell_n+1)-h}$. This shows how to regain the full guessing history.

Next we show how to compute the contributions $\Delta_i(y^{(j)})$ of the entries in the i-th block. To this end, note that the constant substrings $[c \dots c]$ in the reference strings x^c in total contribute exactly $|B_i|$ to the sum $\mathrm{eq}(z, x^0) + \dots + \mathrm{eq}(z, x^k)$. Formally, $\sum_{c=0}^{k-1} \mathrm{eq}([z_{(i-1)s+1} \dots z_{\min\{is, n-1\}}], [c \dots c]) = |B_i|$. Since all other positions of the sampling string y are not changed during the phase, in which we determine the i-th block, we infer that

$$\Delta_i(y^{(j)}) = \mathrm{eq}(z, y^{(j)}) - (\mathrm{eq}(z, x^0) + \dots + \mathrm{eq}(z, x^k) - |B_i|)/k\,.$$

Consequently, in lines 12–14, the algorithm can compute $\Delta_i(y^{(j)})$ for all $j \in [t]$. From this it can infer

$$\mathcal{S}_i^{\text{consistent}} := \{\tilde{z} \in \mathcal{C}^{|B_i|} \mid \forall j \in [t] : \mathrm{eq}(\tilde{z}, \texttt{BLOCK}_i(y^{(j)})) = \Delta_i(y^{(j)})\},$$

the set of possible code segments in B_i. By Theorem 3, the expected size of $\mathcal{S}_i^{\text{consistent}}$ is bounded from above by $1 + 1/|B_i|$. Thus, in lines 12–14 we need an expected number of $1 + 1/|B_i|$ samples w chosen from $\mathcal{S}_i^{\text{consistent}}$ uniformly at random until we find a $y = \texttt{substitute}(y, B_i, w)$ with $\Delta_i(y) = s$ (which implies that the i-th block of y coincides with Carole's secret code). This shows how we determine the entries of the i-th block in an expected total number of $t = O(s/\log s)$ guesses.

When Algorithm 3 executes line 15, all but the last entry of y coincide with Carole's secret code. Hence trying random colors in the n-th position finds the hidden code z with an additional expected number of $k = \Theta(1)$ guesses.

To turn Algorithm 3 into a size-two memory-restricted one, we use the first ℓ_n entries of x to store in binary the iteration counter i, which indicates the index of the block currently being under consideration. This will move the storage space for the guesses and answers by ℓ_n positions to the right. Formally, we define $i(x) := \sum_{h=0}^{\ell_n - 1} 2^h x_{\ell_n - h}$. The inner for loop needs no additional memory to be simulated, because we can learn from $p_1(x)$ how many guesses $q(x)$ have been queried already. More precisely, since storing each guess requires $s + \ell_n + 1$ positions and the first ℓ_n positions are used for indicating the number of already determined entries, we have $q(x) := (p_1(x) - \ell_n)/(s + \ell_n + 1)$.

Lastly, we need to replace the sequential queries in lines 9 and 11 of Algorithm 3 (as this exploits information stored in the program counter). Fortunately, again we can deduce from the memory where we stand. We define a function $\texttt{Part}(y, x)$, which equals 1 if the information of y has been added to the storage string x already and which equals 0 otherwise.

Algorithm 4: A size-two memory-restricted algorithm winning the k-color black answer-peg only Mastermind game in $O(n/\log n)$ guesses. **Remark:** x denotes the unique string in $\mathcal{M}$ with $x_n = 1$ and y denotes the unique string in $\mathcal{M}$ with $y_n = 0$.

1 **Initialization:** Let $\mathcal{M} \leftarrow \emptyset$; // clear memory
2 **if** $\mathcal{M} = \emptyset$ **then**
3 $\quad y \leftarrow [0...0]$; //first reference string
4 $\quad$ Query $\mathrm{eq}(z, y)$ and update $\mathcal{M} \leftarrow \{(y, \mathrm{eq}(z, y))\}$;
5 **else if** $|\mathcal{M}| = 1$ **then**
6 $\quad x \leftarrow [0...0|1]$; //initialization of storage string
7 $\quad$ Query $\mathrm{eq}(z, x)$ and update $\mathcal{M} \leftarrow \mathcal{M} \cup \{(x, \mathrm{eq}(z, x))\}$;
8 **else if** $i(x) < \lceil (n-1)/s \rceil$ **then**
9 $\quad$ **if** $x = [0 \dots 0|1]$ **or** $\Delta_{i(x)}(y) = |B_{i(x)}|$ **then**
10 $\quad\quad x \leftarrow [\mathtt{binary}_{\ell_n}(i(x)+1)|\mathtt{BLOCK}_{i(x)+1}(y)|\mathtt{binary}_{\ell_n}(\mathrm{eq}(z, y))|1|0 \dots 0|1]$; //clear storage string and add first reference string
11 $\quad\quad$ Query $\mathrm{eq}(z, x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z, x))$;
12 $\quad$ **else if** $\mathit{Part}(y, x) = 1$ **and** $q(x) < t + k$ **then**
13 $\quad\quad$ **if** $q(x) < k$ **then** $y \leftarrow \mathtt{substitute}(y, B_{i(x)}, [q(x) \dots q(x)])$; //reference string
14 $\quad\quad$ **else** $y \leftarrow \mathtt{substitute}(y, B_{i(x)}, r)$ where $r \in \mathcal{C}^{|B_{i(x)}|}$ u.a.r.; //random guess
15 $\quad\quad$ Query $\mathrm{eq}(z, y)$ and update $\mathcal{M}$ by replacing $(y, \mathrm{eq}(z, y))$;
16 $\quad$ **else if** $\mathit{Part}(y, x) = 0$ **and** $\Delta_{i(x)}(y) < |B_{i(x)}|$ **then**
17 $\quad\quad x \leftarrow [x_1 \dots x_{p_1(x)}|\mathtt{BLOCK}_{i(x)}(y)|\mathtt{binary}_{\ell_n}(\mathrm{eq}(z, y))|1|0 \dots 0|1]$; //add y's info to x
18 $\quad\quad$ Query $\mathrm{eq}(z, x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z, x))$;
19 $\quad$ **else if** $\mathit{Part}(y, x) = 1$ **and** $q(x) = t + k$ **then**
20 $\quad\quad y \leftarrow \mathtt{substitute}(y, B_{i(x)}, w)$ where $w \in \mathcal{S}^{\text{consistent}}_{i(x)}$ chosen u.a.r.;
21 $\quad\quad$ Query $\mathrm{eq}(z, y)$;
22 $\quad\quad$ **if** $\Delta_{i(x)}(y) = |B_{i(x)}|$ **then** Update $\mathcal{M}$ by replacing $(y, \mathrm{eq}(z, y))$;
23 **else if** $i(x) = \lceil (n-1)/s \rceil$ **then**
24 $\quad y \leftarrow \mathtt{substitute}(y, \{n\}, c)$ where $c \in \mathcal{C} \backslash \{y_n\}$ u.a.r.;
25 $\quad$ Query $\mathrm{eq}(z, y)$;
26 Go to line 2;

That is, we set

$$\mathtt{Part}(y, x) = \begin{cases} 1, & \text{if } \sum_{i=1}^{\ell_n} 2^{i-1} x_{p_1(x)-i} = \mathrm{eq}(z, y) \\ & \quad \text{and } \mathtt{BLOCK}_{i(x)}(y) = [x_{p_1(x)-\ell_n-|B_{i(x)}|} \dots x_{p_1(x)-\ell_n-1}] \\ 0, & \text{otherwise}. \end{cases}$$

Note that $\mathtt{Part}(y, x) = 1$ indicates that the information of y has been stored in x also in the case that our current sample equals the previous one. This is no problem as then the current guess does not give any new information. Hence the use of `Part` modifies the algorithm to sample t random guesses without immediate repitition. Note that the probability to sample the same string $r \in \mathcal{C}^{|B_{i(x)}|}$ twice in a row is at most $1/2$ (if the last block consists only of one position and $k = 2$) and is typically much smaller. Hence, occurrences of this event have no influence on the asymptotic number of guesses needed to win the game.

With these modifications, Algorithm 3 becomes the truly size-two memory-restricted Algorithm 4.

4 Memory of Size One: Proof of Theorem 1

Compared to the situation in Section 3, Paul faces two additional challenges in the size-one memory-restricted setting. The obvious one is that he has less memory available, in particular, after a large part of the code has been determined and needs to be stored. The more subtle one is that he cannot any longer query a search point and then store whatever is worth storing in the second memory cell. With one memory cell, all he can do is to guess a new string and keep or forget it.

Before we give a proof of Theorem 1, let us discuss a linear time winning strategy, i.e., a strategy that allows Paul to find Carole's secret code in a linear expected number of guesses using one memory cell only. This linear time strategy will be used in the proof of Theorem 1 to determine the last $\Theta(n/\log n)$ entries of the secret code.

The basic idea of the linear time strategy is to test each position one by one, from left to right. Since we have just one memory cell, we need to indicate in this one string, which entries have been determined already. We do so by keeping all not yet determined entries at one identical value different from the one of the entry determined last. To this end, let us for all $x \in \mathcal{C}^n$ define

$$\mathrm{tn}(x) := \min\{i \in [n] \mid \forall j \in \{i, \dots, n\} : x_j = x_i\},$$

the *tail number of* x. The following lemma describes the linear time strategy.

▶ **Lemma 4.** *Let $x \in \mathcal{C}^n$. Furthermore, let us denote Carole's secret code by $z \in \mathcal{C}^n$. Let us assume that the first $\mathrm{tn}(x) - 1$ entries of z have been determined (i.e., Carole can no longer change the entries of $[z_1 \dots z_{\mathrm{tn}(x)-1}]$). Further assume that $x_i = z_i$ for all $i < \mathrm{tn}(x)$ and that $\mathcal{M} = \{(x, \mathrm{eq}(z, x))\}$ is the current content of the memory cell.*

There is a size-one memory-restricted guessing procedure `LinAlg` *that—even if Carole plays a devil's strategy—after an expected constant number of successive calls modifies the memory such that the string y now in the memory satisfies $y_i = z_i$ for all $i \leq \mathrm{tn}(x)$ and $\mathrm{tn}(y) = \mathrm{tn}(x) + 1$. Every call of* `LinAlg` *requires only one guess.*

Interestingly, for the definition of `LinAlg`, we need to distinguish between the cases of $k = 2$ and $k \geq 3$ colors, as certain arguments exploit particular properties of these cases. We claim that Algorithm 5 certifies Lemma 4 for $k = 2$ colors. Here we denote, for all $i \in [n]$, by e_i^n the i-th unit vector of length n.

Algorithm 5: Routine `LinAlg` for $k = 2$ colors

Assumption: The string $x \in \{0,1\}^n$ in the memory satisfies $\mathrm{tn}(x) < n$ and $x_i = z_i$ for all $i < \mathrm{tn}(x)$;
Sample $y \in \{x \oplus e^n_{\mathrm{tn}(x)}, x \oplus \sum_{i=\mathrm{tn}(x)+1}^{n} e_i^n\}$ uniformly at random;
Query $\mathrm{eq}(z, y)$;
if $y = x \oplus e^n_{\mathrm{tn}(x)}$ **then**
  if $\mathrm{eq}(z, y) > \mathrm{eq}(z, x)$ **then** $\mathcal{M} \leftarrow \{(y, \mathrm{eq}(z, y))\}$;
else
  if $\mathrm{eq}(z, x) + \mathrm{eq}(z, y) = n + \mathrm{tn}(x)$ **then** $\mathcal{M} \leftarrow \{(y, \mathrm{eq}(z, y))\}$;

Algorithm 6: A size-one memory algorithm winning the k-color Mastermind game in $O(n/\log n)$ guesses.

1 **Initialization:** Let $\mathcal{M} \leftarrow \emptyset$;
2 **if** $\mathcal{M} = \emptyset$ **then**
3 $\quad x \leftarrow [0 \dots 0]$;
4 $\quad$ Query $\mathrm{eq}(z, x)$ and update $\mathcal{M} \leftarrow \{(x, \mathrm{eq}(z, x))\}$;
5 **if** $\exists c \in \mathcal{C} : \mathit{suffix}(x) = [cc] \wedge \mathrm{tn}(x) \leq \ell$ **then**
6 $\quad$ `LinAlg`; //find the first ℓ entries $[z_1 \dots z_\ell]$
7 **else if** $\exists c \in \mathcal{C} : \mathit{suffix}(x) = [cc] \wedge \mathrm{tn}(x) = \ell + 1$ **then**
8 $\quad x \leftarrow [\underbrace{0 \dots 0}_{\ell} | \underbrace{0 \dots 0}_{bs} | x_1 \dots x_\ell | \underbrace{0 \quad \dots \quad 0}_{n-(2\ell+bs+\ell_s+2)} | \mathtt{binary}_{\ell_s}(1) | 01]$; //copy prefix (which coincides with the hidden code)
9 $\quad$ Query $\mathrm{eq}(z, x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z, x))$;
10 **else if** $\mathit{suffix}(x) = [01] \wedge i(x) \leq b \wedge q(x) < t + k$ **then**
11 $\quad$ Apply `Sampling`;
12 **else if** $\mathit{suffix}(x) = [01] \wedge i(x) \leq b \wedge q(x) = t + k$ **then**
13 $\quad$ Apply `OptimizeBlock`;
14 **else if** $\mathit{suffix}(x) = [01] \wedge i(x) = b + 1$ **then**
15 $\quad x \leftarrow [x_{\ell+bs+s+1} \dots x_{2\ell+bs+s+1} | x_{\ell+1} \dots x_{\ell+bs} | c \dots c]$ with $c \in \mathcal{C} \backslash \{x_{\ell+bs}\}$ u.a.r.;
16 $\quad$ Query $\mathrm{eq}(z, x)$ and update $\mathcal{M}$ by replacing $(x, \mathrm{eq}(z, x))$; //prepares x for `LinAlg`
17 **else if** $\exists c \in \mathcal{C} : \mathit{suffix}(x) = [cc] \wedge \ell + bs < \mathrm{tn}(x) \leq n - 2$ **then**
18 $\quad$ `LinAlg`;
19 **else if** $\exists c \in \mathcal{C} : \mathit{suffix}(x) = [cc] \wedge \mathrm{tn}(x) = n - 1$ **then**
20 $\quad$ Sample $y \in \{[x_1 \dots x_{n-2} | p] \mid p \in \mathcal{C}^2\} \backslash \{x\}$ uniformly at random;
21 $\quad$ Query $\mathrm{eq}(z, y)$;
22 $\quad$ **if** $\mathrm{eq}(z, y) = n$ **then** $\mathcal{M} \leftarrow \{(y, \mathrm{eq}(z, y))\}$; //hidden code found
23 Go to line 2;

A proof of Lemma 4 and the algorithm `LinAlg` for $k \geq 3$ colors can be found in [4]. Building on `LinAlg`, we can now present Paul's strategy for the one-memory setting and thus prove Theorem 1.

The very rough overview of Paul's strategy is the following. He determines the first $n - \Theta(n/\log n)$ positions using random guessing, where he manages to store the random substrings and Carole's answers in the yet undetermined part of his one string in the memory. As in the proof of Theorem 2, he does so by iteratively determining blocks of length $s := \lceil \sqrt{n} \rceil$. Then, using the linear time strategy from Lemma 4, he determines the missing entries in $O(n/\log n)$ guesses.

To distinguish between the sampling and the linear time phase, Paul uses the last two entries $\mathtt{suffix}(x) := [x_{n-1}x_n]$ of his string x. He has $\mathtt{suffix}(x) = [01]$, when he is in the random guessing phase, and he uses $\mathtt{suffix}(x) = [cc]$ for some $c \in \mathcal{C}$ to indicate that he applies calls to `LinAlg`. Once Paul has determined all but the last two entries (visible from $\mathrm{tn}(x) = n - 1$), he simply needs to sample uniformly at random from the set of all $k^2 - 1$ remaining possible strings. This clearly finds z in a constant expected number of queries.

The total expected number of guesses can be bounded by

$$\overbrace{\tfrac{n-2}{s}(1-\Theta(\log^{-1} n))}^{\text{number of blocks determined in phase 1}} \overbrace{O(\tfrac{s}{\log s})}^{\text{queries needed to determine any such block}} + \overbrace{O(\tfrac{n}{\log n})}^{\text{queries needed in the 2nd phase}} + \overbrace{O(1)}^{\text{phase 3}} = O(\tfrac{n}{\log n}).$$

A non-trivial part is the random guessing phase. As in the proof of Theorem 2, after guessing $t+k$ strings, we want to be able to regain the full guessing history. If we simply stored the random substring and Carole's reply in some unused part of x, then this changed memory would influence Carole's next answer and we would be unable to deduce information on the next guess from it. We solve this difficulty as follows. We store Carole's latest reply (i.e., value $\mathrm{eq}(z,x)$ currently in the memory) and we sample new (random) substrings for the current block at the same time. Here we store the value $\mathrm{eq}(z,x)$ in a part of x for which we know the entries of Carole's hidden code. By this, we can separate in Carole's next answer the influence of the just stored information from the one of the random guess. The precise description of this `Sampling` substrategy can be found in [4].

To gain this storage space where we know the hidden code, we start with another phase, Phase 0, in which we apply the `LinAlg` procedure $O(\log n)$ times until we found the first $\ell := \ell_n + 1$ positions of z (cf. Lemma 4).

The pseudo-code for the size-one memory-restricted strategy winning the Mastermind game with k colors in $O(n/\log n)$ guesses is given in Algorithm 6. Similar to the notation in the proof of Theorem 2, we denote for any $h \in [0..n]$ the binary encoding of length ℓ_n by $\texttt{binary}_{\ell_n}(h)$ and we denote the binary encoding of length $\ell_s := \lceil \log s \rceil + 1$ by $\texttt{binary}_{\ell_s}(h)$. The current block of interest $i(x)$ is encoded in positions $\{n-\ell_s-1,\dots,n-2\}$, i.e., we have $i(x) := \sum_{h=0}^{\ell_s-1} 2^h x_{n-2-h}$ and $B_{i(x)} := \{\ell + (i(x)-1)s + 1, \dots, \ell + i(x)s\}$. The total number of blocks which we determine via random guessing is $b := \lfloor \frac{n-2}{s}(1 - \frac{K}{\log n}) \rfloor$ for some suitable large constant K. The number of random guesses for each block is $t := \lceil (2+\varepsilon)\frac{s(1+2\log k)}{\log s - \log k} \rceil$ for some arbitrarily small constant $\varepsilon > 0$. Lastly, the actual number of already sampled guesses for block $B_{i(x)}$ is denoted by $q(x)$. As discussed in the proof of Theorem 2, $q(x)$ can be computed via the largest position $p_1 < n-2-\ell_s$ with $x_{p_1} = 1$. Note that the `Sampling` routine described above implicitly updates the counter $q(x)$ by changing the p_1 value. The `OptimizeBlock` routine determines $\texttt{BLOCK}_{i(x)}(z)$, stores it in $B_{i(x)}$ and increases the block counter $i(x)$ by one.

To end this proof sketch, let us show that our memory cell has enough storage capacity to store all $t+k$ substrings, the values $\texttt{binary}_{\ell_n}(\mathrm{eq}(z,x))$, and the values $\texttt{binary}_{\ell_s}(\Delta_{i(v)}(v))$. We have $n-2-(2\ell+bs) = n - n(1-K/\log n) - O(\log n) = Kn/\log n - O(\log n)$ positions for storing information and the total number of positions needed for storing the $t+k$ sample informations is

$$(t+k)(s+O(\log n)) = \Theta(n/\log n) + o(n/\log n) < Kn/\log n - O(\log n)$$

for constant, but sufficiently large K.

Acknowledgment

Carola Winzen is a recipient of the Google Europe Fellowship in Randomized Algorithms. This research is supported in part by this Google Fellowship.

References

1 Gautham Anil and R. Paul Wiegand. Black-box search by elimination of fitness functions. In *Proceedings of the 10th ACM Workshop on Foundations of Genetic Algorithms (FOGA'09)*, pages 67–78. ACM, 2009.

2 Zhixiang Chen, Carlos Cunha, and Steven Homer. Finding a hidden code by asking questions. In *Proceedings of the 2nd Annual International Conference on Computing and Combinatorics (COCOON'96)*, pages 50–55. Springer, 1996.

3 Vasek Chvátal. Mastermind. *Combinatorica*, 3:325–329, 1983.

4 Benjamin Doerr and Carola Winzen. Playing mastermind with constant-size memory. *CoRR*, abs/1110.3619, 2011.

5 Stefan Droste, Thomas Jansen, and Ingo Wegener. Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory of Computing Systems*, 39:525–544, 2006.

6 Paul Erdős and Alfréd Rényi. On two problems of information theory. *Magyar Tud. Akad. Mat. Kutató Int. Közl.*, 8:229–243, 1963.

7 Michael T. Goodrich. On the algorithmic complexity of the mastermind game with black-peg results. *Information Processing Letters*, 109:675–678, 2009.

8 Donald E. Knuth. The computer as a master mind. *Journal of Recreational Mathematics*, 9:1–5, 1977.

9 Jeff Stuckman and Guo-Qiang Zhang. Mastermind is NP-complete. *INFOCOMP Journal of Computer Science*, 5:25–28, 2006.

Polynomial-time Isomorphism Test for Groups with Abelian Sylow Towers

László Babai[1] and Youming Qiao[2]

1 **Department of Computer Science, the University of Chicago**
laci@cs.uchicago.edu
2 **Institute for Interdisciplinary Information Sciences, Tsinghua University**
jimmyqiao86@gmail.com

Abstract

We consider the problem of testing isomorphism of groups of order n given by Cayley tables. The trivial $n^{\log n}$ bound on the time complexity for the general case has not been improved over the past four decades. Recently, Babai et al. (following Babai et al. in SODA 2011) presented a polynomial-time algorithm for groups without abelian normal subgroups, which suggests solvable groups as the hard case for group isomorphism problem. Extending recent work by Le Gall (STACS 2009) and Qiao et al. (STACS 2011), in this paper we design a polynomial-time algorithm to test isomorphism for the largest class of solvable groups yet, namely *groups with abelian Sylow towers*, defined as follows. A group G is said to possess a Sylow tower, if there exists a normal series where each quotient is isomorphic to a Sylow subgroup of G. A group has an abelian Sylow tower if it has a Sylow tower and all its Sylow subgroups are abelian. In fact, we are able to compute the coset of isomorphisms of groups formed as coprime extensions of an abelian group, by a group whose automorphism group is known. The mathematical tools required include representation theory, Wedderburn's theorem on semisimple algebras, and M. E. Harris's 1980 work on p'-automorphisms of abelian p-groups. We use tools from the theory of permutation group algorithms, and develop an algorithm for a parameterized version of the graph-isomorphism-hard setwise stabilizer problem, which may be of independent interest.

1998 ACM Subject Classification I.1.2 Algorithms

Keywords and phrases polynomial-time algorithm, group isomorphism, solvable group

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.453

1 Introduction

We consider the Group Isomorphism problem when groups are given by their Cayley tables. We design a polynomial-time algorithm to test isomorphism for the largest class of solvable groups yet, namely *groups with abelian Sylow towers*, defined as follows. A group G is said to possess a Sylow tower, if there exists a normal series where each quotient is isomorphic to a Sylow subgroup of G. A group has an abelian Sylow tower if it has a Sylow tower and all its Sylow subgroups are abelian. If two groups G and G^* are isomorphic, the set of isomorphisms is a coset of $\mathrm{Aut}(G)$ in $\mathrm{Sym}(G \cup G^*)$, thus can be represented by a set of generators of $\mathrm{Aut}(G)$ and an isomorphism between G and G^*.

▶ **Theorem 1.1.** *There is a polynomial-time algorithm that decides isomorphisms between two groups with abelian Sylow towers, when the groups are given by Cayley tables. If the groups are isomorphic, the algorithm computes the coset of their isomorphisms.*

Let us sketch the current state of the Group Isomorphism problem. For the general case, a straightforward $n^{\log n + O(1)}$ algorithm has been known for about four decades (cf. [25]) and

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 453–464
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

has not been improved. While isomorphism of abelian groups can be tested in linear time according to Kavitha [21] (improving Savage's $O(n^2)$ [29] and Vikas's $O(n \log n)$ [33]), the next natural target to consider, p-groups of class 2, turns out to be currently intractable; nothing significantly better than the trivial $n^{\log n}$ bound is known. We note that p-groups are solvable. Partly for this reason, Babai et al. consider semisimple groups, i.e, groups without abelian normal subgroups (in certain sense the opposite of solvable groups), and present a polynomial-time algorithm for this class in [6] (building on [5]). This work thus amplifies the significance of the solvable case; every group has a solvable normal subgroup such that the quotient is semisimple.

Recently, some progress has been made for some classes of solvable groups, notably for those with at least one abelian normal Hall subgroup.

To explain, let us recall some definitions. For a group G, let N be a normal subgroup, and H a subgroup of G. We say that G is a *semidirect product* of N and H, denoted as $G = N \rtimes H$, if $NH = G$ and $N \cap H = \{\mathrm{id}\}$. H is called a *complement* of N in G. A subgroup is a *Hall subgroup* if its order is coprime to its index. The Schur-Zassenhaus theorem states that a normal Hall subgroup always has a complement. In [26], Qiao, Sarma and Tang present efficient algorithms for groups with an abelian normal Hall subgroup, assuming the complement is either a group with a bounded number of generators, or an elementary abelian group. [26] builds on a technique by Le Gall [13] which allows the normal Hall subgroup to go from elementary abelian to abelian, and an algorithmic result by Babai [4] to test equivalence of linear codes.

In [26], linear representation theory of finite groups was brought to bear on the Group Isomorphism problem, and the reason that the case when the complement is elementary abelian can be solved is mainly because the representations of elementary abelian groups is well-known and can be combined with the algorithmic result on code equivalence. Even the case when the complement is abelian, was not known. The main contribution of this paper is to combine ideas from permutation group algorithms with several mathematical results to compute in polynomial time the automorphism group of the semidirect product $G = A \rtimes H$, where A is an abelian normal Hall subgroup, assuming $\mathrm{Aut}(H)$ is known. The following result is an inductive tool used to prove Theorem 1.1. It can also be interpreted that we are able to test isomorphism of groups formed as coprime extensions of an abelian group, by a group whose automorphism group is known.

▶ **Theorem 1.2.** *Let A be an abelian group, and H a group of order coprime to $|A|$. Suppose two groups G and G^* both can be decomposed as $A \rtimes H$. Then there is a polynomial-time algorithm that computes the coset of isomorphisms between G and G^*, when the groups are given by Cayley tables, and $\mathrm{Aut}(H)$ is given by a list of generators.*

Successive applications of Theorem 1.2 along the Sylow tower gives our main result (Section 3.1).

We briefly indicate the techniques involved. The main object of study is the action of the automorphism group of H on the set of all linear representations of H up to equivalence of representations. Basic facts of representation theory allow us to interpret this action as a parameterized version of the *string G-isomorphism problem* (G a permutation group acting on the set of indices), the starting point of Luks' polynomial-time algorithm to test isomorphism of graphs with bounded degree [22]. Thus we introduce the following parameterized version of the setwise stabilizer problem: given $P \leq \mathrm{Sym}(\Omega)$ and $\Delta \subseteq \Omega$, compute $P_{\{\Delta\}} = \{\pi \in P \mid \Delta^\pi = \Delta\}$ in time $2^{|\Delta|} \cdot \mathrm{poly}(|\Omega|)$. We solve this problem by adapting Luks's dynamic programming technique for hypergraph isomorphism [24]. Finally, we need to generalize an argument by Le Gall [13] which reduces the case of abelian normal Hall

subgroups to elementary abelian. The generalization allows the complement to be an arbitrary group, rather than just a cyclic group. Of particular relevance is the work by M. E. Harris on p'-automorphisms of abelian p-groups ([15], [16]). We also need an old result by A. Ranum [27] on matrix representations of the automorphisms of abelian groups, and Wedderburn's classical theory of semisimple algebras (cf. Chapter 5 in [1]).

It has long been believed that p-groups or equivalently, nilpotent groups (as they are direct product of p-groups) represent the hard cases for Group Isomorphism problem. Our results do not change this perception since all nilpotent groups with abelian Sylow subgroups are abelian.

1.1 Related work and the organization of the paper

We mention some group theory literature related to groups with abelian Sylow towers. A Sylow tower of a group has been used in Chapter 7.6 in Gorenstein's Finite Groups [14]. A group is called an A-group if all its Sylow subgroups are abelian. An A-group is not necessarily solvable, e.g. A_5, while a group with an abelian Sylow tower is solvable. Properties of A-groups have been studied, cf. e.g. [31], [20], and Chapter 12 in [9]. In particular, in [9], the number of non-isomorphic solvable A-groups of order $\leq n$ is proved to be $n^{O(\log n)}$. On the other hand, in [26], the number of non-isomorphic groups with abelian Sylow towers of order $\leq n$ is shown to be $n^{\Omega(\log n)}$. Both Sylow towers and A-groups are discussed at length in Huppert's classical monograph [19].

The construction of finite groups of a given order has been studied by group theorists. To achieve this goal, criteria of isomorphism have been developed. The content of Section 4.2 is adapted from Taunt's work on constructing A-groups [32]. In [8], Besche, Eick and O'Brien surveyed the construction of finite groups of order at most 2000. In particular, in [8, Section 3.3], coprime split extensions are considered, and a practical algorithm, due to Eick, for listing all groups formed by coprime split extensions is presented. Work along these lines often reflects brilliant insights and can be a potential guidance to polynomial-time algorithms (as Taunt's work in our case), but they do not (at least not directly) address the time complexity of isomorphism testing, since their concern is to list all groups of certain order up to isomorphism in practice.

Several recent papers are related to or motivated by the group isomorphism problem. The works on groups without abelian normal subgroups ([5] and [6]), and groups with abelian normal Hall subgroups ([13] and [26]) have been mentioned in the introduction. p-groups of class 2 are generally believed to be the barrier for group isomorphism problem, and in this regard, recent work by Wilson [34, 35] on the structure of p-groups is noteworthy. From the complexity-theoretic perspective, we note that in [11], Chattopadhyay, Torán, and Wagner show that graph isomorphism has no AC^0-reduction to group isomorphism.

Given a permutation group $P \leq \mathrm{Sym}(\Omega)$ and a subset $\Delta \subseteq \Omega$ of the permutation domain, the setwise stabilizer problem asks to compute $P_{\{\Delta\}} = \{\pi \in P \mid \Delta^\pi = \Delta\}$. Our algorithm runs in $2^{|\Delta|} \cdot \mathrm{poly}(|\Omega|)$. It is inspired by Luks's simply-exponential time algorithm for hypergraph isomorphism [24]. Generalizing the special case of the graph isomorphism problem introduced by Babai in 1979 [3] to hypergraphs, Arvind et al. [2] consider the vertex-colored hypergraph isomorphism problem, where isomorphisms are required to preserve the colors and the color-classes have bounded size. They give a polynomial-time algorithm for this case. As a tool, they consider another parameterized version of the setwise stabilizer problem; the parameter is the size t of some P-stable set containing Δ. They presented an algorithm with the same running time as ours, with t in the place of $|\Delta|$. Our parameter "subsumes" theirs (as in our case Δ is not required to be contained in some small P-stable set), and the algorithms are different.

The rest of the paper is organized as follows. We present the preliminaries in Section 2. In Section 3, we explain the reduction of Theorem 1.1 to Theorem 1.2, and give an outline of the proof of Theorem 1.2. In particular, in Section 3.2, we first reduce the original problem of isomorphism computation (Problem 1) to Problems 2 and 3. This reduction is detailed in Section 4. In Section 5, we present solutions to Problems 2 and 3 for the special case when the normal group is elementary abelian; in this section, we establish new connections to permutation group algorithms. Finally in Section 6 we indicate how general case of Problems 2 and 3 reduces to elementary abelian case.

2 Preliminaries

2.1 General group theory

Groups are finite in this paper. Here we present a brief account of the concepts used, and introduce notation. For a group G, if $T \subseteq G$ generates G we write $G = \langle T \rangle$. We write $H \leq G$ for H being a subgroup of G. An *inner automorphism* of a group G is the *conjugation* by an element $g \in G$, i. e., the map $\iota_g : x \mapsto x^g : g^{-1}xg$ $(x \in G)$. A subgroup $N \leq G$ is normal if it is invariant under all inner automorphisms of G, and N is a characteristic subgroup of G if it is invariant under all automorphisms of G. A subgroup is a *Hall subgroup* if its order and its index are coprime. Given a group G and $N \lhd G$, G is the group extension of N by G/N. If N is a normal Hall subgroup, then G is the coprime extension of N by G/N. Given a semidirect product decomposition $G = N \rtimes H$, the conjugation action of H on N gives a homomorphism $\alpha : H \to \mathrm{Aut}(N)$. We denote this situation by $G = N \rtimes_\alpha H$. In the language of extension theory of groups, G is called the split extension of N by H. The well-known Schur-Zassenhaus theorem asserts that a normal Hall subgroup always has a complement. That is, all coprime extensions split. In [26] it is observed that its proof (e.g. Section 9 in [1]) is constructive, giving an efficient algorithm to compute a specific complement.

▶ **Theorem 2.1** (Algorithmic Schur-Zassenhaus theorem, cf. [26]). *Let G be a finite group. Given a normal Hall subgroup $N \lhd G$, there exists $H \leq G$ such that $G = N \rtimes H$, and such an H can be computed in polynomial time. If H and H^* are two complements of N, then H and H^* are conjugates.*

A (right) coset of H in G containing $g \in G$ is $Hg = \{hg \mid h \in H\}$. Given two groups G and G^*, the set of their isomorphisms is denoted by $\mathrm{Iso}(G, G^*)$. Given a group G and $\phi \in \mathrm{Aut}(G)$, we write the action of ϕ in the exponent, that is for $g \in G$, g^ϕ is the image of g under ϕ.

Given a finite set Ω, $\mathrm{Sym}(\Omega)$ denotes the symmetric group consisting of all permutations of Ω. A permutation group acting on Ω is a subgroup of $\mathrm{Sym}(\Omega)$. Given $\pi \in \mathrm{Sym}(\Omega)$ and $a \in \Omega$, the image of a under π is denoted by a^π. For $A \subseteq \Omega$, $A^\pi = \{a^\pi \mid a \in A\}$. Given a permutation group $P \leq \mathrm{Sym}(\Omega)$ and $x, y \in \Omega$, denote $P_{x \to y} = \{\pi \in P \mid x^\pi = y\}$. For a pair of subsets A, $B \subseteq \Omega$ of the same size, denote $P_{A \to B} = \{\pi \in P \mid A^\pi = B\}$. Let $\mathbb{F}$ be a field. Given a vector space V over $\mathbb{F}$, the general linear group $\mathrm{GL}(V)$ consists of all non-singular linear transformations of V.

By the fundamental theorem of finite abelian groups, a finite abelian group is isomorphic to a direct product of cyclic groups of prime power orders. Formally, let A be an abelian group, then there exists a direct product decomposition of A as $A = \langle e_1 \rangle \times \langle e_2 \rangle \times \cdots \times \langle e_n \rangle$, where $e_i \in A$ has order $p_i^{k_i}$, s.t. $p_1 \leq p_2 \leq \cdots \leq p_n$, and if $p_i = p_{i+1}$, then $k_i \leq k_{i+1}$, for all i. This decomposition is called the primary decomposition of A, and the tuple $(e_1, \ldots, e_n)$

forms a basis of A. An elementary abelian group is $\mathbb{Z}_p^n$, where p is a prime. Its automorphism group is isomorphic to $\mathrm{GL}(n,p)$. The set of generators of $\mathrm{GL}(n,p)$ described in Theorem 4.12 from [1] suffices for our use, and a suitable generalization can give a set of generators for $\mathrm{Aut}(A_p)$ where A_p is an abelian p-group.

2.2 Linear representations of finite groups

A (linear) representation of a finite group G over a field $\mathbb{F}$ is a homomorphism $G \to \mathrm{GL}(V)$ where V is a vector space over $\mathbb{F}$. In this paper, a representation of a group is over the field $\mathbb{F}_p$ of prime order p which is coprime to the order of the group. Given representations $\alpha, \beta : G \to \mathrm{GL}(n,p)$, $\hom(\alpha,\beta) := \{\phi \in M(\mathbb{F}_p, n) \mid \alpha(g)\phi = \phi\beta(g), \forall g \in G\}$, and $\mathrm{Iso}(\alpha,\beta) = \{\phi \in \hom(\alpha,\beta), \phi \text{ non-singular}\}$. If $\psi, \psi' \in \hom(\alpha,\beta)$, then $\psi + \psi' \in \hom(\alpha,\beta)$. This shows that $\hom(\alpha,\beta)$ is an algebra. α and β are *equivalent*, denoted as $\alpha \sim \beta$, if $\mathrm{Iso}(\alpha,\beta)$ is not empty. An invariant subspace U of $\alpha : G \to \mathrm{GL}(V)$ is a subspace of V such that $\forall g \in G$, $\alpha(g)(U) = U$. The restriction of α to U, $\alpha|_U$ is called a sub-representation of α. $\vec{0}$ and V are called trivial invariant subspaces. A representation without non-trivial invariant subspaces is an *irreducible representation*.

Schur's lemma states that for two irreducible representations α and β, if they are not equivalent, then $\hom(\alpha,\beta) = \{0\}$. If they are equivalent, then $\hom(\alpha,\beta)$ is a skew field. In the equivalence case, if α, β are over $\mathbb{F}_p$, Wedderburn's "little" theorem ensures $\hom(\alpha,\beta)$ to be a field, and thus $\mathrm{Iso}(\alpha,\beta) = \hom(\alpha,\beta)^\times$. Given a representation $\phi : G \to \mathrm{GL}(V)$, if $V = U \oplus W$, where U and W are invariant subspaces, then ϕ is called the direct sum of $\phi|_U$ and $\phi|_W$. A representation is completely reducible, if it is a direct sum of irreducible sub-representations. Maschke's theorem states that any representation $\phi : G \to \mathrm{GL}(n,\mathbb{F})$ where $\mathrm{char}(\mathbb{F}) \nmid |G|$ is completely reducible.

Let $\mathrm{Rep}(G,\mathbb{F})$ denote the set of linear representations of G over $\mathbb{F}$ *up to equivalence*, and $\mathrm{Irr}(G,\mathbb{F})$ to denote the set of all irreducible representations of G over $\mathbb{F}$ *up to equivalence*. An action of $\mathrm{Aut}(G)$ on $\mathrm{Rep}(G,\mathbb{F})$ can be defined. For $\phi \in \mathrm{Aut}(G)$ and $\alpha \in \mathrm{Rep}(G,\mathbb{F})$, $\alpha^\phi(g) = \alpha(g^{\phi^{-1}})$, $\forall g \in G$. The set $\mathrm{Irr}(G,\mathbb{F})$ is a stable set under this action.

Next, we list some facts about equivalence of representations. For a representation $\alpha : G \to \mathrm{GL}(n,\mathbb{F})$, viewing $\alpha(g)$ as a matrix and taking the trace, we get the *character of* α, denoted as $\chi_\alpha : G \to \mathbb{F}$. Two representations over a field of characteristic that does not divide $|G|$ are equivalent if and only if their characters are the same. Given a completely reducible representation ϕ and an irreducible representation τ of a group G, the multiplicity of τ in ϕ is the number of occurrences of τ (up to equivalence) in the direct sum decomposition of ϕ. Comparing the multiplicities of irreducibles provides another criterion for equivalence of completely reducible representations.

2.3 Representing groups in algorithms

At different stages of the algorithm, we will be concerned with abstract groups, permutation groups, and linear groups, so we summarize their representations here. Abstract groups are given by their Cayley tables. Operations in subgroups and quotient groups are easy by referring to the tables. If a group is of some particular type (e.g., cyclic or elementary abelian), we may use their natural representations implicitly (e.g. $\mathbb{Z}_m$, $\mathbb{Z}_p^n$). For a linear group in $\mathrm{GL}(n,p)$, we assume all matrices in this group are given explicitly. A permutation group $P \leq \mathrm{Sym}(\Omega)$ is represented by a list of generators. A coset Pr in $\mathrm{Sym}(\Omega)$ is represented by a set of generators T of P and a coset representative r', denoted as $\langle T \rangle r'$. For an abstract

group G, $\mathrm{Aut}(G)$ can be viewed as a subgroup of $\mathrm{Sym}(G)$, and $\mathrm{Iso}(G, G^*)$ as a coset of $\mathrm{Aut}(G)$ in $\mathrm{Sym}(G \cup G^*)$.

We will be concerned with representing a coset of a direct product of groups. Given groups G and H, for a coset $L = Kr$ in $G \times H$, we denote by $K_G \subseteq G$ and $K_H \subseteq H$ the projections of K on the first and second coordinates, resp. For $h \in K_H$, let $K_G(h) = \{g \in G \mid (g, h) \in K\}$. It is a coset of $K_G(\mathrm{id}_H)$. We can then represent Kr as follows.

▶ Claim 1. Given $\langle T \rangle = K_H$, $\langle S \rangle = K_G$, and for every $h \in T$ some $r_h \in K_G(h)$, $\langle T \cup S \cup \{r_h \mid h \in T\}\rangle r = Kr$.

2.4 Algorithms for permutation groups and linear representations

Algorithms for permutation groups given by a list of generators have been studied and analyzed, cf., e.g. [23] and [30]. By Sims's "sifting" procedure, from any set of generators of $P \leq \mathrm{Sym}([n])$, a set of generators of size $O(n)$ can be computed. The next proposition follows from Sims's method.

▶ Proposition 1 (Point-transporter algorithm, cf. [23], Proposition 3.9). Given $\langle S \rangle = P \leq \mathrm{Sym}(\Omega)$, $|\Omega| = n$, $x, y \in \Omega$, $P_{x \to y}$ can be computed in $\mathrm{poly}(n, |S|)$.

We describe some algorithmic tasks about linear representations and their solutions. As in our setting, linear representations are given by listing all matrices, the solutions to these tasks will mostly follow from the definitions. We remark that the tasks for representations over finite fields such as the decomposition into irreducible components, and comparing if two irreducible representations are equivalent can be done much more efficiently, even when only generators are given (cf. [28] and [18, Chapter 7]). To compute a basis of $\hom(\alpha, \beta)$ (as an algebra) can be viewed as computing the kernel of a system of linear equations by writing out the linear equations by definition of $\hom(\alpha, \beta)$.

▶ Proposition 2. Given $\alpha, \beta \in \mathrm{Rep}(G, \mathbb{F}_p)$, a basis of $\hom(\alpha, \beta)$ can be computed in time $\mathrm{poly}(|G|, n)$.

▶ Proposition 3. (cf. [26, Section 2]) Given a representation $\phi : G \to \mathrm{GL}(V)$, its irreducible components can be listed in time $O(\dim(V)^2 \cdot |V| \cdot |G|)$.

We can use characters to tell the type of an irreducible representation. It follows from Proposition 3 that we can compute the multiplicities of irreducible representations.

3 About the main theorems

3.1 Reduction of Theorem 1.1 to Theorem 1.2

We explain the reduction of Theorem 1.1 to Theorem 1.2. Let G and G^* be the groups whose isomorphisms we wish to compute. Given a group G, it is not hard to show that if G possesses an abelian Sylow tower, then the Sylow tower can be computed in polynomial time. Two Sylow towers of G and H are *compatible*, if the ith factors, counting from bottom of the tower, are of the same order. Thus we first decide if G and G^* have compatible abelian Sylow towers. If they do not, it is decided that they are not isomorphic. For two compatible towers $\{\mathrm{id}\} = G_0 \lhd G_1 \lhd \cdots \lhd G_\ell = G$ and $\{\mathrm{id}\} = G_0^* \lhd G_1^* \lhd \cdots \lhd G_\ell^* = G^*$, as the base case, $G/G_{\ell-1}$ and $G^*/G^*_{\ell-1}$ are both abelian so their isomorphisms can be computed (e.g. use [10] to compute the primary decomposition, and then a set of generators of the automorphism group of an abelian p-group can be described). Given $\mathrm{Iso}(G/G_i, G^*/G_i^*)$, to compute $\mathrm{Iso}(G/G_{i-1}, G^*/G^*_{i-1})$ we can apply Theorem 1.2, as G_i/G_{i-1} is a normal Sylow

p-subgroup of G/G_{i-1} for some prime p. (Note that every two factors in the Sylow tower are of coprime orders.)

3.2 Outline of the proof of Theorem 1.2

Recall that Theorem 1.2 requires to solve the following problem. It is legitimate to consider just abelian p-groups as if the normal Hall subgroup is abelian, we can form an abelian Sylow tower of the normal Hall subgroup and apply the idea of iterating along the Sylow tower as in Section 3.1.

▶ Problem 1 (Isomorphism computing). Given a group H and $\mathrm{Aut}(H)$, let p be a prime not dividing $|H|$, and A_p an abelian p-group. Given homomorphisms $\alpha : H \to \mathrm{Aut}(A_p)$ and $\beta : H \to \mathrm{Aut}(A_p)$, compute $\mathrm{Iso}(A_p \rtimes_\alpha H, A_p \rtimes_\beta H)$, in time $\mathrm{poly}(|A_p|, |H|)$.

We introduce some further definitions. We call a homomorphism from H to $\mathrm{Aut}(D)$ a *(generalized) representation* of H on D. Usually the group D is from some specified group class. For example, when D is an elementary abelian group $\mathbb{F}_p^n$, then $\mathrm{Aut}(D) \cong \mathrm{GL}(n, p)$ and we are dealing with the representation theory over $\mathbb{F}_p$. In our more general setting, the groups D will be abelian p-groups ($p \nmid |H|$). In analogy with linear representations, two (generalized) representations $\alpha : H \to \mathrm{Aut}(D)$ and $\beta : H \to \mathrm{Aut}(D)$, α and β are called *equivalent*, if there exists $\psi \in \mathrm{Aut}(D)$, such that for every $h \in H$, $\alpha(h)^\psi = \beta(h)$. We denote this by $\alpha \sim_\psi \beta$, and $\alpha \sim \beta$ if ψ is clear from context. We also denote the set of the representations of H over D, *up to equivalence* by $\mathrm{Rep}(H, D)$. Then an action of $\mathrm{Aut}(H)$ on $\mathrm{Rep}(H, D)$ can be defined as follows: for $\phi \in \mathrm{Aut}(H)$ and $\alpha \in \mathrm{Rep}(H, D)$, $\alpha^\phi(h) = \alpha(h^{\phi^{-1}})$. For an action of a group H on the domain Ω, and two points $x, y \in \Omega$, we denote $H_{x \to y} = \{h \in H \mid x^h = y\}$. Given these definitions, in Section 4, Problem 1 breaks up to the following two problems.

▶ Problem 2 (Representation-transporting automorphisms). Given a group H and $\mathrm{Aut}(H)$, let p be a prime not dividing $|H|$, and A_p an abelian p-group. Given homomorphisms $\alpha, \beta : H \to \mathrm{Aut}(A_p)$, compute $\mathrm{Aut}(H, \alpha, \beta) := \mathrm{Aut}(H)_{\alpha \to \beta} = \{\phi \in \mathrm{Aut}(H) \mid \alpha^\phi \sim \beta\}$, in time $\mathrm{poly}(|A_p|, |H|)$.

▶ Problem 3 (Intertwining automorphisms). Given a group H and $\mathrm{Aut}(H)$, let p be a prime not dividing $|H|$, and A_p an abelian p-group. Given homomorphisms $\alpha, \beta : H \to \mathrm{Aut}(A_p)$, such that $\alpha \sim \beta$, compute $\mathrm{Aut}(A_p, \alpha \sim \beta) := \{\psi \in \mathrm{Aut}(A_p) \mid \alpha \sim_\psi \beta\}$, in time $\mathrm{poly}(|A_p|, |H|)$.

The cases when A_p is elementary abelian are of great importance. We will solve the elementary abelian case in Section 5, and reduce the abelian case to the elementary abelian case in Section 6.

4 Breaking Problem 1 to Problem 2 and Problem 3

Recall that Problem 1 requires computing the coset of isomorphisms between $A_p \rtimes_\alpha H$ and $A_p \rtimes_\beta H$, where $p \nmid |H|$. Problem 2 requires computing $\mathrm{Aut}(H, \alpha, \beta) = \{\phi \in \mathrm{Aut}(H) \mid \alpha^\phi \sim \beta\}$, and Problem 3 requires computing $\mathrm{Aut}(A_p, \alpha \sim \beta) = \{\psi \in \mathrm{Aut}(A_p) \mid \alpha \sim_\psi \beta\}$. In this section we consider the more general situation when the normal subgroup is a Hall subgroup, not necessarily abelian.

4.1 Reducing to isomorphisms preserving a decomposition

Given a group $G = N \rtimes_\alpha H$, N a normal Hall subgroup, denote automorphisms in $\mathrm{Aut}(G)$ that send H to H by $\mathrm{Aut}^*(G)$. We will show that the structure of $\mathrm{Aut}(G)$ is essentially

determined by $\mathrm{Aut}^*(G)$. First note that a normal Hall subgroup is characteristic. Then by Schur-Zassenhaus theorem, any $\phi \in \mathrm{Aut}(G)$ sends N to N and H to a conjugate of H. For $g \in G$, writing $g = nh$ where $n \in N$ and $h \in H$, it is clear that $H^g = H^{n'}$, for $n' = n^h$. Thus all conjugates of H are $\{H^n \mid n \in N\}$. For $n \in N$, let $\mathrm{Aut}(G, n) = \{\phi \in \mathrm{Aut}(G) \mid \phi(H) = H^n\}$, then we have $\mathrm{Aut}(G) = \cup_{n \in N} \mathrm{Aut}(G, n)$.

▶ Claim 2. $\phi \in \mathrm{Aut}^*(G)$ if and only if $\phi \circ \iota_n \in \mathrm{Aut}(G, n)$.

As for Problem 1, Claim 2 then tells us that it is enough to focus on the isomorphisms that send H to H, as others can be recovered by composing with an inner automorphism.

4.2 The structure of isomorphisms preserving a decomposition

The content of this subsection is adapted from [32] by Taunt (cf. Theorem 3.3 in [32]). In the following, suppose we are given $G = N \rtimes_\alpha H$ and $G^* = N \rtimes_\beta H$, N is normal Hall in G and G^*. The set of isomorphisms between G and G^* preserving the decomposition, denoted by $\mathrm{Iso}^*(G, G^*)$, is $\{\phi \in \mathrm{Iso}(G, G^*) \mid \phi(N) = N, \phi(H) = H\}$. We will develop the characterization of isomorphisms in $\mathrm{Iso}^*(G, G^*)$ by examining their restrictions to the normal subgroups and to the complements.

▶ **Definition 4.1.** (ν, η) for $\nu \in \mathrm{Aut}(N)$, $\eta \in \mathrm{Aut}(H)$ is a *compatible pair* w.r.t. α and β, if for all $h \in H$, $\alpha(h) = \nu^{-1} \circ \beta(\eta(h)) \circ \nu$.

Let the set of all compatible pairs w.r.t. G and G^* be $\mathrm{Com}(G, G^*) \subseteq \mathrm{Aut}(N) \times \mathrm{Aut}(H)$. We write $\mathrm{Com}(G)$ for $\mathrm{Com}(G, G)$. It can be verified that $\mathrm{Com}(G) \leq \mathrm{Aut}(N) \times \mathrm{Aut}(H)$, and $\mathrm{Com}(G, G^*)$ is a coset of $\mathrm{Com}(G)$. We then show that $\mathrm{Com}(G, G^*)$ captures $\mathrm{Iso}^*(G, G^*)$.

▶ **Theorem 4.2.** *For $G = N \rtimes_\alpha H$, $G^* = N \rtimes_\beta H$, there is a bijection between* $\mathrm{Iso}^*(G, G^*)$ *and* $\mathrm{Com}(G, G^*)$. *In particular,* $\mathrm{Aut}^*(G) \cong \mathrm{Com}(G)$.

4.3 Reducing Problem 1 to Problem 2 and Problem 3

We now can see how Problem 1 breaks into Problem 2 and Problem 3. Suppose we are given $G = N \rtimes_\alpha H$ and $G^* = N \rtimes_\beta H$. By discussion in Section 4.1 we know it is enough to consider isomorphisms sending H to H, that is $\mathrm{Iso}^*(G, G^*)$. Then by Theorem 4.2 we need to compute $\mathrm{Com}(G, G^*)$, which is a coset in $\mathrm{Aut}(N) \times \mathrm{Aut}(H)$. We first consider the projection of $\mathrm{Com}(G, G^*)$ on $\mathrm{Aut}(H)$. Then $\eta \in \mathrm{Aut}(H)$ is in the projection if and only if there exists $\nu \in \mathrm{Aut}(N)$ such that (ν, η) is a compatible pair, which by Definition 4.1 just induces equivalence of representations α^η, $\beta \in \mathrm{Rep}(H, N)$. Thus we get Problem 2. Suppose we are given $\langle T \rangle = \mathrm{Aut}(H, \alpha, \beta)$, to compute a set of generators for $\mathrm{Com}(G, G^*)$ Claim 1 shows that we need to compute for every $\eta \in T$ a set of generators for $\{\nu \in \mathrm{Aut}(N) \mid (\nu, \eta)$ is a compatible pair$\}$, which is essentially Problem 3.

5 When the normal subgroup is elementary abelian

5.1 Solving Problem 2 when the normal subgroup is elementary abelian

5.1.1 Parameterized setwise stabilizer problem

We first introduce the algorithmic tool that will be used. Given a permutation group $P \leq \mathrm{Sym}(\Omega)$ and $\Delta \subseteq \Omega$, Setwise stabilizer problem asks to compute $P_{\{\Delta\}} := P_{\Delta \to \Delta}$. It is one of Luks's equivalence class above Graph Isomorphism [23], and in [7], it is shown to be in NP$\cap$coAM, thus not expected to be NP-complete unless the polynomial hierarchy collapses.

Inspired by Luks's dynamic programming procedure for hypergraph isomorphism [24], our algorithms takes into account the size of the set Δ. In fact we will solve the parameterized set-transporter problem.

▶ Proposition 4 (Parameterized set-transporter problem). For $P \leq \mathrm{Sym}(\Omega)$, $A, B \subseteq \Omega$, $|\Omega| = n$ and $|A| = |B| = k$, there is an algorithm in time $2^k \cdot \mathrm{poly}(n)$ that computes $P_{A\to B}$.

Proof. Put an order to elements in A as $\{x_1, \ldots, x_k\}$, and let $A_i = \{x_1, \ldots, x_i\}$, for $i \in [k]$. We will build a dynamic programming table indexed by (A_i, C) for $C \subseteq B$ of size i to store $P_{A_i\to C}$. To start, for every $b \in B$, $P_{x_1\to b}$ can be computed by the point-transporter algorithm in Proposition 1. Now assume that for every $C' \subseteq B$ of the same size $\ell - 1$, we have computed $P_{A_{\ell-1}\to C'}$. For A_ℓ, $C \subseteq B$ of size ℓ, we can compute $P_{A_\ell\to C}$ by the equation $P_{A_\ell\to C} = \bigcup_{b\in C}(P_{A_{\ell-1}\to C\setminus\{b\}})_{x_\ell\to b}$, where $P_{A_{\ell-1}\to C\setminus\{b\}}$ can be read from the table, and $(P_{A_{\ell-1}\to C\setminus\{b\}})_{x_\ell\to b}$ can be computed by Proposition 1. After taking union over $b \in C$, apply Sims's method to get a small set of generators. To analyze the running time, the number of table entries is bounded by 2^k. For each entry we apply point transporter algorithm and sifting procedure for at most k times, which runs in time $\mathrm{poly}(n)$. ◀

Another classical problem in permutation group algorithms is the string P-isomorphism problem, where P is a permutation group acting on the indices of the strings. This problem is the starting point of Luks's polynomial time algorithm to test isomorphism of graphs of bounded degree [22]. For a permutation group $P \leq \mathrm{Sym}(\Omega)$ and a function $f : \Omega \to [\ell]$, the action of $\pi \in P$ on f, denoted as f^π, is defined by $f^\pi(x) = f(x^{\pi^{-1}})$. $[\ell]$ can be considered as a set of colors to be assigned to points in the permutation domain. Now given two functions $f_1, f_2 : \Omega \to [\ell]$, the problem asks to compute the coset $\mathrm{Iso}_P(f_1, f_2) := \{\pi \in P \mid f_1^\pi = f_2\}$. We consider the parameterized version of this problem, where the sum of sizes of *all but one* colors is bounded.

▶ **Corollary 5.1** (Parameterized string P-isomorphism problem). *For a permutation group $P \leq \mathrm{Sym}(\Omega)$, $|\Omega| = n$, and two functions $f_1, f_2 : \Omega \to [\ell]$, such that for $j \in [2]$, $\sum_{i\in[\ell-1]} |f_j^{-1}(i)| \leq k$. Then $\mathrm{Iso}_P(f_1, f_2)$ can be computed in time $2^k \cdot \mathrm{poly}(n)$.*

5.1.2 Reducing Problem 2 to parameterized string P-isomorphism problem

Recall that Problem 2, when the normal subgroup is elementary abelian, requires to compute for two linear representations $\alpha, \beta : H \to \mathrm{GL}(n, p)$, $\mathrm{Aut}(H, \alpha, \beta) = \{\phi \in H \mid \alpha^\phi \sim \beta\}$. Let Ω be the set of all irreducible representations of H. α induces $\alpha' : \Omega \to \{0, 1, \ldots, n\}$ by assigning an irreducible representation $\omega \in \Omega$ to its multiplicity in α. As the total number of irreducibles in α is bounded by the dimension of the representation, $\sum_{i\in[n]} |\alpha'^{-1}(i)| \leq |\Omega| \leq n$. Similarly we have $\beta' : \Omega \to \{0, 1, \ldots, n\}$.

▶ **Lemma 5.2.** $\mathrm{Aut}(H, \alpha, \beta) = \mathrm{Iso}_{\mathrm{Aut}(H)}(\alpha', \beta')$.

Lemma 5.2 shows the reduction from Problem 2 in elementary abelian case to string P-isomorphism problem to be executed. It can be achieved in time polynomial in $\mathrm{poly}(p^n, |H|)$ in conjunction with the algorithms for linear representations presented in Section 2.4. We leave the details to the full version.

▶ **Theorem 5.3.** *For a group H, suppose $\mathrm{Aut}(H)$ is given by generators, and representations $\alpha, \beta : H \to \mathrm{GL}(n, p)$ are given by listing the matrices. Then $\mathrm{Aut}(H, \alpha, \beta) = \{\phi \in \mathrm{Aut}(H) \mid \alpha^\phi \sim \beta\}$ can be computed in time $2^n \cdot \mathrm{poly}(|H|)$.*

5.2 Solving Problem 3 when the normal subgroup is elementary abelian

In Problem 3, when the normal subgroup is elementary abelian, we are given $\alpha, \beta : H \to \mathrm{GL}(n,p)$ such that $\alpha \sim \beta$, and we need to compute those $\psi \in \mathrm{GL}(n,p)$ inducing equivalence between α and β, that is $\mathrm{Iso}(\alpha,\beta)$. We will use $\mathrm{End}(\alpha)$ and $\mathrm{Aut}(\alpha)$ to denote $\hom(\alpha,\alpha)$ and $\mathrm{Iso}(\alpha,\alpha)$. If α and β are irreducible representations over $\mathbb{F}_p$, as discussed in Section 2.2, Schur's lemma states that the $\hom(\alpha,\beta)$ is an extension field of $\mathbb{F}_p$, and $\mathrm{Iso}(\alpha,\beta)$ is the multiplicative group of $\hom(\alpha,\beta)$. Suppose then $\hom(\alpha,\beta)$ is isomorphic to $\mathbb{F}_q$, where $q = p^m$. Since $\mathbb{F}_p^n$ is a $\mathbb{F}_q$-module, $m \mid n$. By Proposition 2, $\hom(\alpha,\beta)$ can be computed and listed in time $\mathrm{poly}(|H|, p^n)$.[1] Given α and β, we first use Proposition 3 to get irreducible components. Then group them by isomorphic types, using the character to distinguish them. As α and β are equivalent, we can assume that α and β are decomposed, and irreducible components grouped as $\mathrm{P} = \mathrm{P}_1 \oplus \mathrm{P}_2 \oplus \cdots \oplus \mathrm{P}_r$, where P_i is the direct sum of k_i copies of ρ_i, and ρ_i is irreducible. By the discussion above, we can list $\mathrm{Aut}(\rho_i) = \mathbb{F}_{q_i}^{\times}$, $q_i = p^{m_i}$. Then we need to use Wedderburn's theory on the structure of semisimple algebras.[2]

▶ **Lemma 5.4** (Lemma 12 and Lemma 13 in [1]). *$\mathrm{End}(\mathrm{P}) \cong \mathrm{End}(\mathrm{P}_1) \oplus \cdots \oplus \mathrm{End}(\mathrm{P}_r)$, and $\mathrm{End}(\mathrm{P}_i) \cong M_{k_i}(\mathbb{F}_{q_i})$, where: $q_i = p^{m_i}$ for some m_i; k_i is the number of copies of ρ_i in P_i.*

Given this lemma, note that automorphisms of P are the invertible elements in $\mathrm{End}(\mathrm{P})$.

▶ Proposition 5. $\mathrm{Aut}(\mathrm{P}) \cong \mathrm{Aut}(\mathrm{P}_1) \oplus \cdots \oplus \mathrm{Aut}(\mathrm{P}_r)$, and $\mathrm{Aut}(\mathrm{P}_i) \cong \mathrm{GL}(n_i, q_i)$, where $q_i = p^{m_i}$ for some m_i, and k_i is the number of copies of ρ_i in P_i.

Finally we recall that a set of generators of $\mathrm{GL}(n_i, q_i)$ can be described easily. We also need to store the change-of-basis matrix when we decompose representations.

6 When the normal subgroup is an abelian p-group

In this section we show that for Problem 2 and Problem 3, the abelian normal Hall subgroup can be reduced to the elementary abelian case. We exhibit the main lemma and indicate the reduction to be executed, while proofs can be found in the full version.

We describe some concepts that are generally useful for the study of p-group, following [14]. For a group G, the *Frattini subgroup* $\Phi(G)$ is the intersection of maximal subgroups, and $G/\Phi(G)$ is called the Frattini factor group of G. For a prime p, the *p-core* $O_p(G)$ is the unique largest normal p-subgroup of G. For a p-group P, it is well-known that its Frattini factor group $P/\Phi(P)$ is elementary abelian. An abelian p-group is *homocyclic*, if its primary decomposition consists of factors of the same order. We will use a slightly improved main technical lemma in [16], the proof of which is put into appendix. (The original lemma deals with the case when B is homocyclic.)

▶ **Lemma 6.1** ([16]). *Let $B = B_1 \times B_2 \times \cdots \times B_k$ be an abelian p-group, where B_i's are the homocyclic components of B of exponent p^{r_i} and order $p^{r_i n_i}$. Denote $A := \mathrm{Aut}(B)$, and $O := O_p(A)$.*

1. *$\mathrm{Aut}(B_i/\Phi(B_i)) \cong \mathrm{GL}(n_i, p)$, $A/O \cong \prod_{i\in[k]} \mathrm{Aut}(B_i/\Phi(B_i))$;*
2. *Let X be a p'-subgroup of A/O. For $i = 1, 2$, $g_i : X \to \mathrm{Aut}(B)$ is a monomorphism such that x and $g_i(x)$ induce the same element of A/O for all $x \in X$. Then there exists $t \in O$ such that $g_2(x) = t^{-1} g_1(x) t$ for all $x \in X$.*

[1] As a field $\mathbb{F}$ is a simple algebra over its prime field, any $\mathbb{F}$-module is a direct sum of some copies of $\mathbb{F}$.
[2] An algebra A is semisimple if any A-module is a direct sum of simple modules. The group algebra of G, $\mathbb{F}G$ is semisimple if $\mathrm{char}(\mathbb{F}) \nmid |G|$. cf. Chapter 5 in [1].

Let the notations as in Lemma 6.1, and $\Lambda_p : A \to A/O$ be the canonical epimorphism. As $A/O \cong \prod_i \mathrm{GL}(n_i, p)$, for $\alpha : H \to A$, $\Lambda_p \circ \alpha$ maps H to $\prod_i \mathrm{GL}(n_i, p)$. Recall that $\mathrm{Aut}(H, \alpha, \beta) = \{\phi \in \mathrm{Aut}(H) \mid \alpha^\phi \sim \beta\}$. The following corollary is an immediate consequence of Lemma 6.1.

▶ **Corollary 6.2.** $\mathrm{Aut}(H, \alpha, \beta) = \mathrm{Aut}(H, \Lambda_p \circ \alpha, \Lambda_p \circ \beta)$.

The above corollary indicates that we need to compute Λ_p as the reduction. From the algorithmic point of view we need to be able to manipulate explicitly the automorphism group of an abelian group. To achieve that Ranum's work on automorphisms of abelian groups ([27], cf. also [17] and [13]) will be crucial. This reduction is first proposed by Le Gall in [13], and he proved for for the case when the complement is cyclic. Here we just proved that the same reduction in [13] can be applied to arbitrary complements. We note that the authors in [26] overlooked the fact that Le Gall's result was proved only for cyclic complements, and used for the situation when the complement is elementary abelian. Finally, Ranum's work also enables the reduction from Problem 3 to A_p being elementary abelian, which can be viewed as solving a system of linear Diophantine equations (cf. [12]) and then a set of generators can be recovered.

Acknowledgement

Youming would like to thank J.L. Alperin and J.B. Wilson for several helpful discussions. Part of the work was done when Youming was visiting the University of Chicago and (then) Microsoft Research India, and he would also like to thank Laci Babai, Neeraj Kayal and Satya Lokam for their warm host. László Babai's work is supported in part by NSF Grant CCF-1017781. Youming Qiao's work is supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61061130540, 61073174.

References

1 J.L. Alperin and R.B. Bell. *Groups and representations*. Graduate texts in mathematics. Springer, 1995.

2 Vikraman Arvind, Bireswar Das, Johannes Köbler, and Seinosuke Toda. Colored hypergraph isomorphism is fixed parameter tractable. In *FSTTCS*, pages 327–337, 2010.

3 László Babai. Monte-Carlo algorithms in graph isomorphism testing. Technical Report 79-10, Univ. de Montréal, Dép. de mathématiques et de statistique, 1979.

4 László Babai. Equivalence of linear codes. Manuscript, 2010. See [5].

5 László Babai, Paolo Codenotti, Joshua A. Grochow, and Youming Qiao. Code equivalence and group isomorphism. In *SODA*, pages 1395–1408, 2011.

6 László Babai, Paolo Codenotti, and Youming Qiao. Polynomial-time isomorphism test for groups without abelian normal subgroups. Manuscript, 2011.

7 László Babai and Shlomo Moran. Arthur-Merlin games: a randomized proof system, and a hierarchy of complexity class. *J. Comput. Syst. Sci.*, 36:254–276, April 1988.

8 Hans Ulrich Besche, Bettina Eick, and E.A. O'Brien. A millennium project: Constructing small groups. *Intern. J. Alg. and Comput*, 12:2002.

9 S.R. Blackburn, P.M. Neumann, and G. Venkataraman. *Enumeration of finite groups*. Cambridge tracts in mathematics. Cambridge University Press, 2007.

10 J. Buchmann and A. Schmidt. Computing the structure of a finite abelian group. *Mathematics of Computation*, 74(252):2017–2026, 2005.

11 Arkadev Chattopadhyay, Jacobo Torán, and Fabian Wagner. Graph isomorphism is not AC^0 reducible to group isomorphism. In *FSTTCS*, pages 317–326, 2010.

12 Tsu-wu J. Chou and George E. Collins. Algorithms for the solution of systems of linear Diophantine equations. *Siam Journal on Computing*, 11:687–708, 1982.

13 François Le Gall. Efficient isomorphism testing for a class of group extensions. In *STACS*, pages 625–636, 2009.

14 D. Gorenstein. *Finite groups*. AMS Chelsea Publishing Series. American Mathematical Society, 2007.

15 M. E. Harris. On p'-automorphisms of abelian p-groups. *Rocky Mountain J. Math.*, 7:751–752, 1977.

16 M. E. Harris. On p'-automorphisms of abelian p-groups, ii. *Periodica Mathematica Hungarica*, 11:321–323, 1980. 10.1007/BF02107573.

17 Christopher Hillar and Darren Rhea. Automorphisms of finite abelian groups. *Amer. Math. Monthly*, 114(10):917–923, 2007.

18 Derek F. Holt, Bettina Eick, and Eamonn A. O'Brien. *Handbook of computational group theory*. Chapman and Hall/CRC, London, 2005.

19 B. Huppert. *Endliche Gruppen*. Number v. 1 in Endliche Gruppen. Springer, 1967.

20 Noboru Itô. Note on A-groups. *Nagoya Mathematical Journal*, 4:79–81, 1952.

21 T. Kavitha. Linear time algorithms for abelian group isomorphism and related problems. *J. Comput. Syst. Sci.*, 73(6):986–996, 2007.

22 Eugene M. Luks. Isomorphism of graphs of bounded valence can be tested in polynomial time. *J. Comp. Sys. Sci.*, 25:42–65, 1982.

23 Eugene M. Luks. Permutation groups and polynomial-time computation. In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 1993.

24 Eugene M. Luks. Hypergraph isomorphism and structural equivalence of boolean functions. In *Proc. 31st ACM STOC*, pages 652–658. ACM Press, 1999.

25 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.

26 Youming Qiao, Jayalal M. N. Sarma, and Bangsheng Tang. On isomorphism testing of groups with normal Hall subgroups. In *STACS*, pages 567–578, 2011.

27 A. Ranum. The group of classes of congruent matrices with application to the group of isomorphisms of any abelian group. *Transactions of the American Mathematical Society*, 8(1):71–91, 1907.

28 Lajos Rónyai. Computing the structure of finite algebras. *J. Symb. Comput.*, 9(3):355–373, 1990.

29 C. Savage. An $O(n^2)$ algorithm for abelian group isomorphism. Technical report, North Carolina State University, 1980.

30 A. Seress. *Permutation group algorithms*, volume 152 of *Cambridge Tracts in Mathematics*. Cambridge University Press, Cambridge, 2003.

31 D. R. Taunt. On A-groups. *Mathematical Proceedings of the Cambridge Philosophical Society*, 45:24–42, 1949.

32 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.

33 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.

34 James B. Wilson. Decomposing p-groups via jordan algebras. *J. Algebra*, 322:2642–2679, 2009.

35 James B. Wilson. Finding central decompositions of p-groups. *J. Group Theory*, 12:813–830, 2009.

Preemptive and Non-Preemptive Generalized Min Sum Set Cover

Sungjin Im[1], Maxim Sviridenko[2], and Ruben van der Zwaan[3]

1 **Department of Computer Science, University of Illinois, 201 N. Goodwin Ave., Urbana, IL 61801, USA.**
im3@illinois.edu

2 **IBM T.J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598, USA.**
sviri@us.ibm.com

3 **Department of Quantitative Economics, Maastricht University, The Netherlands.**
r.vanderzwaan@maastrichtuniversity.nl

Abstract

In the (non-preemptive) Generalized Min Sum Set Cover Problem, we are given n ground elements and a collection of sets $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ where each set $S_i \in 2^{[n]}$ has a positive requirement $\kappa(S_i)$ that has to be fulfilled. We would like to order all elements to minimize the total (weighted) cover time of all sets. The cover time of a set S_i is defined as the first index j in the ordering such that the first j elements in the ordering contain $\kappa(S_i)$ elements in S_i. This problem was introduced by [1] with interesting motivations in web page ranking and broadcast scheduling. For this problem, constant approximations are known [2, 15].

We study the version where preemption is allowed. The difference is that elements can be fractionally scheduled and a set S is covered in the moment when $\kappa(S)$ amount of elements in S are scheduled. We give a 2-approximation for this preemptive problem. Our linear programming and analysis are completely different from [2, 15]. We also show that any preemptive solution can be transformed into a non-preemptive one by losing a factor of 6.2 in the objective function. As a byproduct, we obtain an improved 12.4-approximation for the non-preemptive problem.

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

Keywords and phrases Set Cover, Approximation, Preemption, Latency, Average cover time

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.465

1 Introduction

The Min Sum Set Cover problem is a minimum latency version of the hitting set problem. We are given as input n elements, $\{1, 2, \ldots, n\} = [n]$ and a collection of sets $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ where each set $S_i \in 2^{[n]}$. The goal is to find a permutation of the elements such that the total sum of (or equivalently average) cover/hitting times of all sets is minimized. For simplicity, we will say that an element e is covered at time slot t or it has cover time $\texttt{cov}(e) = t$ if it is placed in the t-th position in the permutation. The cover time $\texttt{cov}(S_i)$ of a set S_i is defined as $\min_{e \in S_i} \texttt{cov}(e)$ and the goal is to minimize $\sum_{S_i \in \mathcal{S}} \texttt{cov}(S_i)$. For this problem, a simple greedy algorithm is known to achieve an approximation factor 4 [4, 8]. The greedy algorithm iteratively picks the element that hits the most sets that are not yet hit. Also it is known that the problem cannot be approximated within a factor of $4 - \epsilon$ for any $\epsilon > 0$ unless $P = NP$ [8]. A closely related problem known as Min Sum Coloring was studied before in

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 465–476
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

[4, 5] with applications in scheduling. Also the special case of the Min Sum Vertex Cover was used in [6] as a heuristic for speeding up a solver for semidefinite programs.

The Min Latency Set Cover problem is a variant where the cover time is defined as the time where all elements in the set are covered e.g. $\texttt{cov}(S_i) = \max_{e \in S_i} \texttt{cov}(e)$. This problem is in fact equivalent to the precedence-constrained scheduling on a single machine [16], for which various 2-approximation algorithms are known [10, 7, 11]. It was shown that, assuming a variant of the Unique Games Conjecture, unless P=NP there is no $2 - \epsilon$ approximation for any $\epsilon > 0$ [3].

A generalization of the aforementioned problems was introduced by Azar, Gamzu and Yin [1] to provide a better framework for ranking web pages in response to queries that could have multiple intentions. This generalized problem was later named Generalized Min Sum Set Cover [2], and can be stated as follows. Every set S_i has a requirement $\kappa(S_i) \in \{1, 2, \ldots, |S_i|\} = [|S_i|]$. For a permutation of the ground set we define $\texttt{cov}(e)$ as before and S_i is covered at time t if t is the earliest time such that $|\{e \in S_i : \texttt{cov}(e) \leq t\}| \geq \kappa(S_i)$. Again, the goal is to find a permutation of the elements in $[n]$ minimizing $\sum_{S_i \in \mathcal{S}} \texttt{cov}(S_i)$. Azar et al. [1] give a modified greedy algorithm that has a performance guarantee of $O(\ln(\max_{S_i \in \mathcal{S}} \kappa(S_i)))$. The question whether there exists an $O(1)$-approximation was answered affirmatively by Bansal, Gupta and Krishnaswamy [2]. In order to obtain an $O(1)$-approximation, they used a time indexed linear program together with knapsack cover inequalities and gave a clever randomized rounding scheme. Very recently, their approximation ratio of 485 was improved by Skutella and Williamson to 28 via the same LP but a different rounding scheme [15].

In this paper we study the Preemptive Generalized Min Sum Set Cover. Like the Generalized Min Sum Set Cover problem, when $\kappa(S) = |S|$ for all $S \in \mathcal{S}$ it is a special case (and in fact is a equivalent to) single machine scheduling problem with precedence constraints and preemptions: $1|prec, pmtn| \sum w_j C_j$. It is known that preemption does not improve the solution quality for this problem (shown by a simple exchange argument), i.e. the optimal preemptive and non-preemptive schedules have the same optimal value. Hence it follows that there is no $2 - \epsilon$ approximation for any $\epsilon > 0$ assuming a variant of the Unique Games Conjecture and $P \neq NP$ [3].

Preemptive Generalized Min Sum Set Cover is formally defined as follows. Given the ground set of elements $[n]$, sets $\mathcal{S} = \{S_1, S_2, ..., S_m\}$ and requirement $\kappa(U) \in [|U|]$ for each set $U \in \mathcal{S}$, we should fractionally assign elements of the ground set to the interval $[0, n]$. Formally, we define functions $x_e(t) : [0, n] \to \{0, 1\}$ where $x_e(t)$ is the indicator function that denotes whether element e is scheduled at time t such that $\int_{t=0}^{n} x_e(t)\, \mathtt{d}t = 1$ for all $e \in [n]$ and $\sum_{e \in [n]} x_e(t) = 1$ for any time $t \in [0, n]$. Then, the cover time $\texttt{cov}(S)$ of the set S is defined as the earliest time t such that $\int_{\tau=0}^{t} \sum_{e \in S} x_e(\tau)\, \mathtt{d}\tau \geq \kappa(S)$ and the goal is to minimize the sum of cover times over all sets. Note that the cover time $\texttt{cov}(S)$ is not necessarily an integer unlike in the non-preemptive problem.

Our main motivation to study Preemptive Generalized Min Sum Set Cover is the fact that it provides a lower bound for the optimal value of the Generalized Min Sum Set Cover. We decouple finding an approximate solution to the relaxed problem (see Section 2) and the question of the lower bound quality (see Section 3 and Conjecture 1).

Our Results

Our main result is a polynomial time approximation algorithm with performance guarantee of 2 for the Preemptive Generalized Min Sum Set Cover. As we noticed before this result is tight modulo some complexity assumptions [3]. We note that one can easily show that the linear program used in [2, 15] is a valid relaxation for the preemptive problem, thus the

best known approximation for the non-preemptive problem also carries for the preemptive problem as well.

We introduce a *configuration linear program* which completely differs from the linear programming relaxation used in [2, 15]. Interestingly, it is not obvious that our new linear program is a valid relaxation for the preemptive problem, unlike the previous linear program in [2, 15] which can be easily shown to be a valid relaxation for the preemptive (and non-preemptive) problem. Our new LP is provably stronger than the previous LP, for both the preemptive and non-preemptive problems.

Further, we study the "gap" between the preemptive and non-preemptive solutions of the Generalized Min Sum Set Cover Problem, which is of independent interest. With some modifications of the rounding scheme in [15], we show that one can transform any α-approximate preemptive schedule into 6.2α-approximate non-preemptive one. With this transformation, we obtain an 12.4-approximation for the non-preemptive Generalized Min Sum Set Cover Problem, improving upon the previous best 28-approximation by Skutella and Williamson[15]. We conjecture that the gap between optimal preemptive and non-preemptive solutions is precisely two.

All our proofs easily extend to the case where every set S_i has a non-negative weight $w_i \geq 0$ and the objective is to minimize $\sum_{S_i \in \mathcal{S}} w_i \cdot \mathsf{cov}(S_i)$.

Organization

The remainder of this paper is organized as follows. In Section 2 we introduce the configuration linear program $\mathsf{LP}_{\mathsf{primal}}$. First, we prove that our configuration linear program is a valid relaxation for Preemptive Generalized Min Sum Set Cover and that this linear program can be solved in polynomial time. Finally, we design a rounding procedure that results in a randomized 2-approximation (Section 2.4) that can be derandomized. In Section 3 we obtain a transformation from a preemptive schedule to a non-preemptive schedule with a loss of factor 6.2, which immediately implies a 12.4-approximation in expectation to Generalized Min Sum Set Cover. In Section 4 we compare the time indexed linear program in [2, 15] to our own configuration linear program and show our linear program is stronger. Due to space constraints, we omit most parts from Section 3 and 4. We will include the full details and omitted proofs in the full version of this paper.

2 2-Approximation for Preemptive Generalized Min Sum Set Cover

This section is devoted to prove the following theorem.

▶ **Theorem 1.** *There is a randomized polynomial time 2-approximation algorithm for Preemptive Generalized Min Sum Set Cover.*

Throughout this section, for any integer $t \in [n]$, the t-th time slot will be equivalent to the time interval $[t-1, t]$.

2.1 Configuration LP

We write a configuration linear program. For a set $S \in \mathcal{S}$, a valid configuration is an (integral) assignment of elements in S to time slots. More formally, such a map can be described as an injective function $f_S : S \to [n]$. For notational simplicity, we may represent the mapping via a relation $F =_{def} \{(e, f_S(e)) \mid e \in S\}$. Let $\mathcal{F}(S)$ denote the collection of all possible configurations for set S. Let C_S^F denote the completion time t of set S under the configuration

F, i.e. the first time t' such that $|f_S^{-1}([t'])| \geq \kappa(S)$. Let $x_{e,t}$ denote the fraction of element e we schedule in the t-th time slot. The variable y_S^F is used to indicate which configurations S adheres to. For example, if $y_S^F = 1$, it means all elements in S are scheduled following the configuration F.

Our linear program is formulated as follows.

$$\min \sum_{S\in\mathcal{S}} \sum_{F\in\mathcal{F}(S)} C_S^F y_S^F \tag{ILP}$$

$$s.t. \quad \sum_e x_{e,t} = 1 \qquad \forall\, t \in [n] \tag{1}$$

$$\sum_t x_{e,t} = 1 \qquad \forall\, e \in [n] \tag{2}$$

$$\sum_{F\in\mathcal{F}(S)} y_S^F = 1 \qquad \forall\, S \in \mathcal{S} \tag{3}$$

$$\sum_{F\in\mathcal{F}(S),(e,t)\in F} y_S^F = x_{e,t} \qquad \forall\, e,t \in [n], S : e \in S \tag{4}$$

$$x_{e,t} \in \{0,1\} \qquad \forall\, e,t \in [n]$$

$$y_S^F \in \{0,1\} \qquad \forall\, S \in \mathcal{S}, F \in \mathcal{F}(S)$$

The constraints (1) and (2) enforce that exactly one element is scheduled at any time slot and that an element can be scheduled only once over all times. The constraint (3) states that each set S has a unique configuration. Finally, (4) says that if an element e is scheduled at time t, then it must align with the configuration of S.

The relaxation $\mathsf{LP}_{\mathsf{primal}}$ of ILP is then defined as follows.

$$\min \quad \sum_{S\in\mathcal{S}} \sum_{F\in\mathcal{F}(S)} C_S^F y_S^F \tag{$\mathsf{LP}_{\mathsf{primal}}$}$$

$$s.t. \quad \text{Constraints (1),(2),(3) and (4) hold}$$

$$x_{e,t} \geq 0 \qquad \forall\, e,t \in [n]$$

$$y_S^F \geq 0 \qquad \forall\, S \in \mathcal{S}, F \in \mathcal{F}(S)$$

2.2 Validity of the LP

It is easy to verify that $\mathsf{LP}_{\mathsf{primal}}$ is a valid linear programming relaxation for Generalized Min Sum Set Cover. However, it is not obvious that the $\mathsf{LP}_{\mathsf{primal}}$ is indeed a valid relaxation for the preemptive problem. Since we will use two different types of fractional schedules throughout the analysis, we first clearly define/remind those schedules. The first one is a *continuous* schedule that is defined by indicator functions $x_e(t) : [0,n] \to \{0,1\}, e \in [n]$ such that (1) for any $t \in [0,n], \sum_{e\in[n]} x_e(t) = 1$ and (2) for any $e \in [n]$, $\int_{\tau=0}^{n} x_e(\tau)\, \mathrm{d}\tau = 1$. We say that $x_e(t), e \in [n]$ is a feasible schedule if all these conditions are satisfied. Recall that the completion time C_S of each set S is defined by a continuous schedule as the earliest time t such that $\int_{\tau=0}^{t} \sum_{e\in S} x_e(\tau)\, \mathrm{d}\tau \geq \kappa(S)$. The other version of schedule, which is somewhat *discretized*, is defined by $x_{e,t}, e, t \in [n]$ that satisfy (1) $\sum_{e\in[n]} x_{e,t} = 1$, (2) $\sum_{t\in[n]} x_{e,t} = 1$ and (3) $0 \leq x_{e,t} \leq 1$ for any $e, t \in [n]$. When these conditions are satisfied, we will say $x_{e,t}, e, t \in [n]$ is feasible. Note that this discretized version of schedule does not immediately define the completion time of sets. Rather, it is used in $\mathsf{LP}_{\mathsf{primal}}$ as a relaxation of continuous schedules. We show the following theorem.

▶ **Theorem 2.** *Consider any feasible continuous schedule $x_e(t), e \in [n]$. Let C_S denote the completion time of S in this schedule. For any $e, t \in [n]$, let $x_{e,t} =_{def} \int_{\tau=t-1}^{t} x_e(\tau)\, d\tau$. Then $x_{e,t}$ satisfy constraints (1) and (2). Also there exists y-values that satisfy the other constraints (3) and (4) as well and further satisfy*

$$\sum_{S\in\mathcal{S}} \sum_{F\in\mathcal{F}(S)} C_S^F y_S^F \leq \sum_{S\in\mathcal{S}} C_S. \tag{5}$$

The first claim in Theorem 2 that $x_{e,t}$ satisfy constraints (1) and (2) easily follows from the property of continuous schedules and from how $x_{e,t}$ are defined. Due to the space constraints, we defer the proof to the full version of this paper. In fact, it is not difficult to see that there exist y-values that satisfy all constraints (1)-(4). However, we can find an example of y-values satisfying all the constraints but not satisfying the inequality (5) (See full version of this paper). Henceforth, we focus on showing that there exist "good" y-values that also satisfy (5). We will show how to construct a feasible solution y such that the inequality

$$\sum_{F\in\mathcal{F}(S)} C_S^F y_S^F \leq C_S \tag{6}$$

holds for any set $S \in \mathcal{S}$ which will imply the inequality (5). Since setting y_S-values for a specific S does not affect other y-values, we can focus on each $S \in \mathcal{S}$. We will find "good" y_S^F-values that satisfy constraints (3) and (4), and further (6).

To this end, we define two matroids M_1 and M_2 that enforce that any independent set in the intersection of M_1 and M_2 which corresponds to a feasible configuration $F \in \mathcal{F}(S)$. Then we show that the vector $x_{e,t}, e \in S, t \in [n]$ lies in the intersection of the polytopes of the two matroids. Using the fact that such an intersection polytope is integral, we will be able to decompose x into a convex combination of integer points that lie in the intersection of the polytopes of M_1 and M_2. As already mentioned, due to the structure of the matroids, each integer point will correspond to a configuration $F \in \mathcal{F}(S)$. By setting y-values as suggested by the decomposition, we will guarantee that y satisfy constraints (3) and (4). Finally, we will complete the analysis by showing that such y-values satisfy (6) as well. This is enabled by some additional constraints we impose on the matroids. We refer the reader to Chapters 39-41 in [13] for the extensive overview of algorithmic matroid theory.

We begin with defining each of the two matroids M_1 and M_2 which have the same common ground set, $U = \{(e,t) \mid e \in S, t \in [n]\}$ (Recall that we are focusing on each fixed $S \in \mathcal{S}$ separately). We will call (e,t) a pair in order to distinguish it from elements, $[n]$. The first matroid $M_1 = (U, \mathcal{I}(M_1))$ enforces that each element in S can be scheduled in at most one time slot. Formally, the collection $\mathcal{I}(M_1)$ of independent sets of M_1 is defined as follows: $A \in \mathcal{I}(M_1)$ if and only if for any $e \in S$, $|A \cap \{(e,t) \mid t \in [n]\}| \leq 1$. Observe that M_1 is a partition matroid since pairs in U are partitioned based on each common element, and any independent set collects at most one pair from each group. Hence the polytope $P(M_1)$ of M_1 (polymatroid) is defined as follows.

$$\sum_{t\in[n]} x_{e,t} \leq 1 \qquad \forall e \in S \tag{$P(M_1)$}$$

$$x_{e,t} \geq 0 \qquad \forall e \in S, t \in [n]$$

▶ Proposition 3. The vector $x = (x_{e,t})$, $e \in S$, $t \in [n]$ is in the polytope $P(M_1)$. Moreover, $\sum_{e\in S, t\in[n]} x_{e,t} = |S|$, i.e. x belongs to the base polymatroid of M_1.

The second matroid $M_2 = (U, \mathcal{I}(M_2))$ has a more involved structure. It enforces that in each time slot, at most one element in S can be scheduled. Additionally, it enforces that at

most $\kappa(S)$ elements can be scheduled during the first $C-1$ time slots and at most $|S|-\kappa(S)$ elements can be scheduled during the time slots, $C+1, C+2, ..., n$, where C is an integer such that $C-1 < C_S \leq C$. This additional constraints will be crucial in finding "good" y-values. Formally, $A \in \mathcal{I}(M_2)$ if and only if A satisfies

- For each integer time $t \in [n]$, $|A \cap \{(e,t) \mid e \in S\}| \leq 1$.
- $|A \cap \{(e,t) \mid e \in S], 1 \leq t \leq C-1\}| \leq \kappa(S)$.
- $|A \cap \{(e,t) \mid e \in S, C+1 \leq t \leq n\}| \leq |S| - \kappa(S)$.

We observe that $\mathcal{I}(M_2)$ is a laminar matroid: All pairs in U are partitioned into groups with the same time t, and at most one pair can be chosen from each group to be in an independent set. Further, the second and third constraints put a limit on the number of pairs that can be chosen from the groups of time slots $t = 1, 2, ..., C-1$ and from the groups of time slots $t = C+1, C+2, ..., n$, respectively. We define the polymatroid $P(M_2)$ as follows.

$$\sum_{e \in S} x_{e,t} \leq 1 \qquad \forall t \in [n] \qquad (P(M_2))$$

$$\sum_{t=1}^{C-1} \sum_{e \in S} x_{e,t} \leq \kappa(S)$$

$$\sum_{t=C+1}^{n} \sum_{e \in S} x_{e,t} \leq |S| - \kappa(S)$$

$$x_{e,t} \geq 0 \qquad \forall e \in S, t \in [n]$$

▶ Proposition 4. The vector $x = (x_{e,t})$ lies in the polymatroid $P(M_2)$.

It is well known the the intersection of two polymatroids is an integral polytope, i.e. any vertex point is integral. Hence since $(x_{e,t})$ lies in the intersection of two polytopes $P(M_1)$ and $P(M_2)$, it can be decomposed into a linear combination of vertex (hence integer) points in $P(M_1) \cap P(M_2)$. Note that each of such integer points corresponds to an independent set in $\mathcal{I}(M_1) \cap \mathcal{I}(M_2)$, which is of size at most $|S|$ due to the constraints of M_1. In fact, the size must be exactly $|S|$, since $\sum_{e \in S} \sum_{t \in [n]} x_{e,t} = |S|$. By the constraints of M_1 and the first constraints of M_2, we conclude that each of such integer points corresponds to a configuration $F \in \mathcal{F}(S)$. Hence we have shown the following lemma.

▶ **Lemma 5.** *There exist $\mathcal{F}'(S) \subseteq \mathcal{F}(S)$ and positive constants $\theta_S^F, F \in \mathcal{F}'(S)$ that satisfy*

- $\sum_{F \in \mathcal{F}'(S)} \theta_S^F = 1$.
- *For any $e \in S$, $t \in [n]$, $x_{e,t} = \sum_{F \in \mathcal{F}'(S)} \theta_S^F \cdot \mathbf{1}[(e,t) \in F]$.*

where an indicator variable $\mathbf{1}[(e,t) \in F] = 1$ if and only if $(e,t) \in F$.

We let $y_S^F = \theta_S^F$ for all $F \in \mathcal{F}'(S)$ and $y_S^F = 0$ for all $F \in \mathcal{F}(S) \setminus \mathcal{F}'(S)$. Note that x and y satisfy constraints (3) and (4).

It remains to show that y satisfy (6). Now the second and third constraints of M_2 play a crucial role. We make the following observation.

▶ **Lemma 6.** *For any $F \in \mathcal{F}'(S)$ exactly one of the following holds.*

- $|F \cap \{(e,t) \mid e \in S, 1 \leq t \leq C-1\}| = \kappa(S)$.
- $|F \cap \{(e,t) \mid e \in S, 1 \leq t \leq C-1\}| = \kappa(S) - 1$ *and $(e,C) \in F$ for some $e \in S$.*

Proof. Recall that $|F| = |S|$. By the third constraints of M_2, we know that $N_{\geq C+1} =_{def} |F \cap \{(e,t) \mid e \in S, C+1 \leq t \leq n\}| \leq |S| - \kappa(S)$, hence that $N_{\geq C} =_{def} |F \cap \{(e,t) \mid e \in S, C \leq t \leq n\}| \leq |S| - \kappa(S) + 1$. Therefore, we have $N_{\leq C-1} =_{def} |F \cap \{(e,t) \mid e \in S, 1 \leq$

$t \le C-1\}| \ge \kappa(S)-1$. Further, we know $N_{\le C-1} \le \kappa(S)$ from the second constraint of M_2. Thus unless $N_{\le C-1} = \kappa(S)$, it must be the case that $N_{\le C-1} = \kappa(S)-1$. In that case, since $N_{\ge C+1} \le |S| - \kappa(S)$, we conclude that $(e, C) \in F$ for some $e \in S$. ◀

Motivated by the above lemma, we can now prove that our linear program is a valid relaxation for the preemptive version of the problem.

Proof of Theorem 2. Partition $\mathcal{F}'(S)$ into $\mathcal{F}'_1(S)$ and $\mathcal{F}'_2(S)$ by letting $\mathcal{F}'_1(S)$ to denote all $F \in \mathcal{F}'(S)$ that fall in the first case in the Lemma 6 and letting $\mathcal{F}'_2(S) = \mathcal{F}'(S) \setminus \mathcal{F}'_1(S)$. Let $\theta' = \sum_{F \in \mathcal{F}'_2(S)} \theta^F_S$. Note that for any $F \in \mathcal{F}'_1(S)$, $C^F_S \le C-1$ and for any $F \in \mathcal{F}'_2(S)$, $C^F_S = C$. In words, the set S is completed no later than time $C-1$ for $(1-\theta')$ fraction of configurations in $\mathcal{F}'(S)$ and exactly at time C for θ' fraction of configurations in $\mathcal{F}'(S)$. Hence we have that

$$\begin{aligned}\sum_{F\in\mathcal{F}(S)} C^F_S y^F_S &= \sum_{F\in\mathcal{F}'(S)} C^F_S \theta^F_S = \sum_{F\in\mathcal{F}'_1(S)} C^F_S \theta^F_S + \sum_{F\in\mathcal{F}'_2(S)} C^F_S \theta^F_S \\ &\le (1-\theta')(C-1) + \theta' C = C - 1 + \theta' \qquad (7)\end{aligned}$$

Now we focus on upper-bounding θ'. From the definition of C_S and the fact that $\sum_{e\in S} x_e(\tau) \le 1$ for any τ, we know that

$$\begin{aligned}\int_{\tau=0}^{C-1} \sum_{e\in S} x_e(\tau)\,\mathrm{d}\tau &= \int_{\tau=0}^{C_S} \sum_{e\in S} x_e(\tau)\,\mathrm{d}\tau - \int_{\tau=C-1}^{C_S} \sum_{e\in S} x_e(\tau)\,\mathrm{d}\tau \\ &\ge \kappa(S) - (C_S - (C-1)) \qquad (8)\end{aligned}$$

On the other hand, it follows that

$$\begin{aligned}&\int_{\tau=0}^{C-1} \sum_{e\in S} x_e(\tau)\,\mathrm{d}\tau = \sum_{t=1}^{C-1}\sum_{e\in S} x_{e,t} \qquad \text{[By the definition of } x_{e,t}\text{]} \\ =\ &\sum_{t=1}^{C-1}\sum_{e\in S}\sum_{F\in\mathcal{F}'(S)} y^F_S \qquad \text{[From the decomposition of } x \text{ into } y^F_S\text{]} \\ =\ &\sum_{F\in\mathcal{F}'_1(S)} y^F_S \sum_{e\in S}\sum_{t=1}^{C-1} \mathbf{1}[(e,t)\in F] + \sum_{F\in\mathcal{F}'_2(S)} y^F_S \sum_{e\in S}\sum_{t=1}^{C-1} \mathbf{1}[(e,t)\in F] \\ =\ &\sum_{F\in\mathcal{F}'_1(S)} \theta^F_S \cdot \kappa(S) + \sum_{F\in\mathcal{F}'_2(S)} \theta^F_S \cdot (\kappa(S)-1) \\ =\ &(1-\theta')\cdot\kappa(S) + \theta'\cdot(\kappa(S)-1) = \kappa(S) - \theta' \qquad (9)\end{aligned}$$

From (8) and (9), we have $\theta' \le C_S - (C-1)$. By combining this with (7), we complete the proof of Theorem 2. ◀

2.3 Solving the LP

The linear programming relaxation $\mathsf{LP}_{\mathsf{primal}}$ has exponentially many variables. Hence, we solve the dual LP and show there are only polynomially many non-zero variables in the primal LP that achieve the optimal LP value. The dual LP is as follows.

$$\max \sum_{t\in[n]} \alpha_t + \sum_{e\in[n]} \beta_e + \sum_{S\in\mathcal{S}} \gamma_S \qquad (\mathsf{LP}_{\mathsf{dual}})$$

$$s.t. \quad \alpha_t + \beta_e - \sum_{S: e\in S} \delta_{etS} \le 0 \qquad \forall e, t \qquad (10)$$

$$\gamma_S + \sum_{(e,t)\in F} \delta_{etS} \le C^F_S \qquad \forall S\in\mathcal{S}, F\in\mathcal{F}(S) \qquad (11)$$

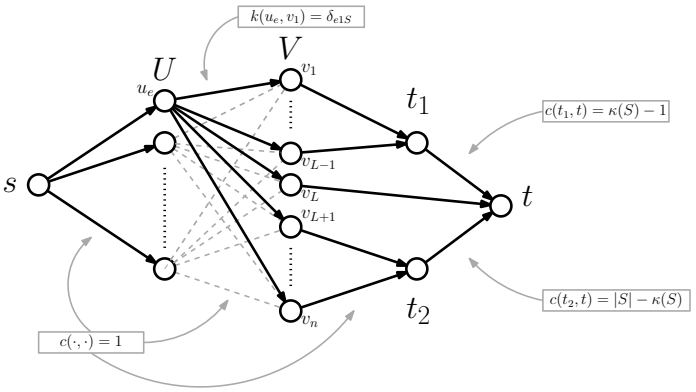


Figure 1 An illustration of the construction of the graph G, in which we want to find a maximum-value flow.

To solve $\mathsf{LP}_{\mathsf{dual}}$ with the ellipsoid algorithm, we need a *separation oracle* for finding a violated constraint (see [9]). Since constraints (10) are easy to verify (there are only n^2 of them), we focus on constraints (11). We need a polynomial time algorithm that given γ_S and δ_{etS}-values, finds (if any) $S \in \mathcal{S}$ and $F \in \mathcal{F}(S)$ that violate constraints (11).

We model this problem as a classical minimum cost s-t flow problem. In this problem, we are given a digraph $G = (V, A)$, a capacity function $c : A \to \mathbb{Q}_+$, a cost function $k : A \to \mathbb{Q}$ and the volume $\phi \in \mathbb{Q}_+$. The goal is to send ϕ amount of flow from the source s to the sink t, i.e. to find an s-t flow f of volume ϕ, subject to capacity constraints $0 \leq f(e) \leq c(e)$ for all $e \in A$ and the standard flow conservation constraints, minimizing the costs $\sum_{e \in A} f(e)k(e)$.

It is known that if the volume ϕ and capacities $c_e, e \in E$ are integral then we can test in polynomial time if there is an s-t flow of volume ϕ. Moreover, if there is such a flow (i.e. there is a feasible solution to the problem) then there is an integral minimum-cost s-t flow, and it can be found in polynomial time (see Chapter 12 in [13]).

We now show how to reduce our separation problem for constraints (11) to the minimum cost s-t flow problem. It will be convenient for us to consider an equivalent maximum cost s-t flow problem where the goal is to maximize the value of the objective function $\sum_{e \in A} f(e)k(e)$.

Fix a set S and an integer $L \in [n]$. We will try to find a violated constraint for the constraints (11) corresponding to the set S and configurations $F \in \mathcal{F}(S)$ with $C_S^F = L$. Create a directed complete bipartite graph $G_L = (U, V, A)$ where part U has vertex u_e for each $e \in S$, part V has vertex v_i for each time slot $i \in [n]$. Arc $a = (u_e, v_i) \in A$ has cost $k(e) = \delta_{eiS}$ and capacity $c(e) = 1$. We augment G_L as follows. We add a source vertex s and connect it to all vertices in U. There are two "intermediate" sinks t_1 and t_2, both connected to the "final" sink t. The vertices $v_1, v_2, ..., v_{L-1}$ in V are connected to t_1 and the vertices $v_{L+1}, v_{L+2}, ..., v_n$ in V are connected to the other intermediate sink t_2. The arcs a between the source s and part U have cost $k(a) = 0$ and capacity $c(a) = 1$. Analogously, all arcs a between part V and intermediate sinks t_1 and t_2 have cost $k(a) = 0$ and capacity $c(a) = 1$. Arcs $a' = (t_1, t)$ and $a'' = (t_2, t)$ have capacities $c(a') = \kappa(S) - 1$ and $c(a'') = |S| - \kappa(S)$ respectively, and all of them have zero costs. The vertex v_L is special and is directly connected to t. The arc (v_L, t) has a unit capacity and zero cost. The goal is to find the s-t flow of volume $\phi = |S|$ of maximum cost. See Figure 1 for an illustration of this construction.

Note that any integral s-t flow f of value $|S|$ in digraph G_L corresponds to a valid configuration F for volume S such that $C_S^F = L$, and vice versa. Hence, if the maximum-cost s-t flow in G_L has cost more than $L - \gamma_S$, the constraint (11) is violated for S and $F \in \mathcal{F}(S)$ that corresponds to the flow. The converse also holds: if the maximum-cost s-t flow has cost less than or equal to $L - \gamma_S$ there is *no* configuration $F \in \mathcal{F}(S)$ with $C_S^F = L$ that violates

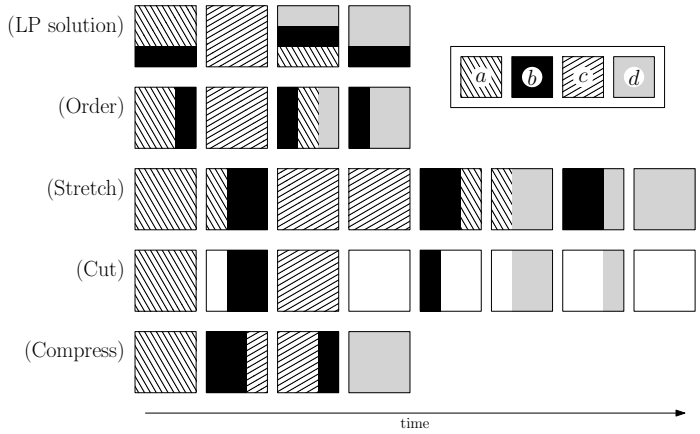


Figure 2 In this example the schedule is stretched by a factor of two e.g. $\lambda = \frac{1}{2}$.

(11). With the help of this separation oracle and classical connection between separation and optimization [9], we can solve $\mathsf{LP}_{\mathsf{dual}}$ in polynomial time.

Then we can optimally solve $\mathsf{LP}_{\mathsf{primal}}$ by focusing only on y_S^F variables that correspond to the constraints that were considered by the ellipsoid method in solving $\mathsf{LP}_{\mathsf{dual}}$. A more formal (and well-known) argument is that the $\mathsf{LP}_{\mathsf{dual}}$ with the subset of constraints considered by the ellipsoid method is a relaxation of the original problem but it has the same optimal solution. The dual of the relaxed problem is $\mathsf{LP}_{\mathsf{primal}}$ restricted to the subset of corresponding variables which by the strong duality theorem has the same optimal value.

2.4 Rounding procedure

Let $x_{e,t}$ and y_S^F be a basic optimal solution of the linear programming relaxation $\mathsf{LP}_{\mathsf{primal}}$. In particular we know that there are at most $2n + m + n^2 m$ non-zero variables (this is the number of constraints (1)-(4)). Let C_S^{LP} denote the completion time of set S in the LP. That is, $C_S^{LP} = \sum_{F \in \mathcal{F}(S)} C_S^F y_S^F$. We create a schedule parameterized by $\lambda \in (0, 1]$, where λ is randomly drawn from $(0, 1]$ according to the density function $f(v) = 2v$.

Create an arbitrary continuous schedule $x_e(t)$, $e \in [n]$, $t \in [0, n]$ from $x_{e,t}, e, t \in [n]$ such that for any $e, t \in [n]$, $\int_{\tau=t-1}^{t} x_e(\tau)\, \mathrm{d}\tau = x_{e,t}$. For example, this can be done by processing each element e for the amount $x_{e,t}$ during the time step t in an arbitrary order between the elements to obtain $x_e(t)$. For notational convenience, let σ denote the continuous schedule $x_e(t)$. The new schedule $\sigma(\lambda)$ is defined as follows. Stretch out the schedule σ by a factor of $\frac{1}{\lambda}$. In other words, map every point τ in time onto τ/λ. For each element e define $\tau_e \in [1, n/\lambda]$ to be the earliest point in time when the element has been processed for one time unit (out of total $1/\lambda$). Leave the machine idle whenever it processes the element e after time τ_e. After repeating this procedure for all elements $e \in [n]$, we shift the whole schedule to the left to eliminate all idle times. The final schedule $\sigma(\lambda)$ has total length n. Let $x_e^{(\lambda)}(t), e \in [n], t \in [0, n]$ be the resulting continuous schedule $\sigma(\lambda)$. Note that similar algorithms were used in scheduling before to design approximation algorithms for various preemptive scheduling problems with total completion time objective [14, 12].

▶ **Example 7.** See Figure 2 for an illustration. Consider an instance with 4 elements $\{a, b, c, d\}$, with the LP solution $x_{a,1} = 2/3$, $x_{b,1} = 1/3$, $x_{c,2} = 1$, $x_{d,3} = 1/3$, $x_{b,3} = 1/3$, $x_{a,3} = 1/3$, $x_{d,4} = 2/3$, $x_{b,4} = 1/3$. Construct a continuous schedule by randomly *ordering* the elements in each time step. For example in time step 3, three elements, a, b, d are scheduled seamlessly, each for $1/3$ time steps. Then *stretch* the whole schedule by a factor two ($\lambda = 1/2$). *cut* out each element after being scheduled by a unit amount. Finally, *compress* the schedule, by shifting everything to the left removing the idle times.

Let $C_S(\lambda)$ denote the completion time of S in the new schedule $\sigma(\lambda)$. Order all configurations $F \in \mathcal{F}(S)$ for $y_S^F > 0$ in non-decreasing order of C_S^F. Let $F_1, F_2, ..., F_k$ be such an ordering. Define $\tilde{C}_s(\lambda) =_{def} C_S^{F_j}$ where $\sum_{i=1}^{j-1} y_S^{F_i} < \lambda$ and $\sum_{i=1}^{j} y_S^{F_i} \geq \lambda$. Let $\mathbf{1}[\phi]$ be an indicator function such that $\mathbf{1}[\phi] = 1$ if and only if ϕ is true and zero otherwise.

▶ **Lemma 8.** *For any $S \in \mathcal{S}$ and $0 < \lambda \leq 1$, $C_S(\lambda) \leq \frac{1}{\lambda} \cdot \tilde{C}_S(\lambda)$.*

Proof. To simplify the proof we assume that there exists j such that $\sum_{1 \leq l \leq j} y_S^{F_l} = \lambda$. Otherwise, let j be the lowest index such that $\sum_{1 \leq l \leq j} y_S^{F_l} > \lambda$, then we define two copies F_j' and F_j'' of configuration F_j, with $y_S^{F_j'} = \lambda - \sum_{1 \leq l \leq j-1} y_S^{F_l}$ and $y_S^{F_j''} = \sum_{1 \leq l \leq j} y_S^{F_l} - \lambda$. Here F_j' and F_j'' are the same configurations as F_j. Now, $\sum_{1 \leq l \leq j-1} (y_S^{F_l}) + y_S^{F_j'} = \lambda$.

We will show the following inequality:

$$\int_{\tau=0}^{\tilde{C}_S(\lambda)/\lambda} \sum_{e \in S} x_e^{(\lambda)}(\tau)\, \mathtt{d}\tau \geq \kappa(S) \tag{12}$$

since it would imply that the completion time $C_S(\lambda)$ of the set S in the schedule $\sigma(\lambda)$ must be no later than $\tilde{C}_S(\lambda)/\lambda$. Since for every $e \in S$ we have $\int_{\tau=0}^{\tilde{C}_S(\lambda)/\lambda} x_e^{(\lambda)}(\tau)\, \mathtt{d}\tau \geq \min\left\{1, \int_{\tau=0}^{\tilde{C}_S(\lambda)} x_e(t)/\lambda\, \mathtt{d}\tau\right\} \geq \min\left\{1, \sum_{t \leq \lfloor \tilde{C}_S(\lambda) \rfloor} x_{e,t}/\lambda\right\}$, and $\tilde{C}_S(\lambda)$ is integral by definition for any $\lambda \in (0, 1]$, it is sufficient to show the inequality

$$\sum_{e \in S} \min\left\{\lambda, \sum_{t \leq \tilde{C}_S(\lambda)} x_{e,t}\right\} \geq \lambda \kappa(S) \tag{13}$$

to derive (12). We now derive the inequality (13).

$$\begin{aligned}
\sum_{e \in S} \min\left\{\lambda, \sum_{t \leq \tilde{C}_S(\lambda)} x_{e,t}\right\} &\geq \sum_{e \in S} \min\left\{\lambda, \sum_{l=1}^{j} y_S^{F_l} \cdot \mathbf{1}[(e,t) \in F_l \text{ for some } t \leq \tilde{C}(\lambda)]\right\} \\
&= \sum_{e \in S} \sum_{l=1}^{j} y_S^{F_l} \cdot \mathbf{1}[(e,t) \in F_l \text{ for some } t \leq \tilde{C}(\lambda)] \\
&= \sum_{l=1}^{j} y_S^{F_l} \sum_{e \in S} \mathbf{1}[(e,t) \in F_l \text{ for some } t \leq \tilde{C}(\lambda)] \\
&\geq \sum_{l=1}^{j} y_S^{F_l} \kappa(S) = \lambda \kappa(S)
\end{aligned}$$

The first inequality follows from constraints (4). The first equality holds because $\sum_{l=1}^{j} y_S^{F_l} = \lambda$. The last inequality holds because for any F_l, $l \leq j$, $C_S^{F_l} \leq \tilde{C}_S(\lambda)$. ◀

The following lemma can be easily shown from the definition of $\tilde{C}_S(\lambda)$.

▶ **Lemma 9.** *For any $S \in \mathcal{S}$, $\int_{\lambda=0}^{1} \tilde{C}_S(\lambda)\, \mathtt{d}\lambda = C_S^{LP}$.*

Proof of Theorem 1. By Theorem 2, $\mathsf{LP}_{\mathsf{primal}}$ is a valid relaxation, and we now show how to round to obtain a 2-approximation (in expectation).

$$\begin{aligned}
\mathbb{E}[C_S(\lambda)] &= \int_{\lambda=0}^{1} C_S(\lambda) \cdot 2\lambda\, \mathtt{d}\lambda \qquad \text{[By definition]} \\
&\leq \int_{\lambda=0}^{1} \frac{1}{\lambda} \cdot \tilde{C}_S(\lambda) \cdot 2\lambda\, \mathtt{d}\lambda \qquad \text{[By Lemma 8]} \\
&= 2 \int_{\lambda=0}^{1} \tilde{C}_S(\lambda)\, \mathtt{d}\lambda \quad = 2C_S^{LP} \qquad \text{[By Lemma 9]}
\end{aligned}$$

◀

We shortly indicate how our approximation algorithm can be derandomized. The function $\tilde{C}_S(\lambda)$ is a piecewise constant function, with at most a polynomial number of pieces since there are at most polynomially many non-zero variables y_S^F for each S. This implies that there are at most polynomially many "interesting" λ-values that we need to consider, among which at least one that gives the desired approximation ratio.

3 Gap between Preemptive and Non-preemptive Schedules

In this section, we study the lower bound quality of the preemptive problem for the non-preemptive problem. The goal is to show that there exists a small gap between the preemptive and non-preemptive solutions (schedules) of Generalized Min Sum Set Cover. Note that if we show a way to convert any given preemptive schedule into a non-preemptive one losing a factor of η, it would immediately obtain a 2η-approximation algorithm for the non-preemptive Generalized Min Sum Set Cover.

Our scheme for transforming a preemptive schedule into non-preemptive one is similar to the one by Skutella and Williamson [15]. We obtain a better gap by utilizing several additional tricks *and* starting from a preemptive schedule. Formally we prove the following theorem, of which the proof is deferred to the full version of this paper.

▶ **Theorem 10.** *Given a preemptive schedule with cost C, then there exists a non preemptive schedule with expected cost at most $6.2C$. Furthermore, this transformation can be done in polynomial time.*

Combining Theorem 1 and Theorem 10 we derive

▶ **Theorem 11.** *There exists a polynomial time 12.4-approximation algorithm for Generalized Min Sum Set Cover.*

We have shown an upper bound on the gap of 6.2, and any gap lower than 2 would result in an approximation factor strictly less than 4 for the non-preemptive problem, which is impossible unless P=NP [8]. We believe that our gap is not tight. In fact, we make the following bold conjecture:

▶ Conjecture 1. Given a preemptive schedule with cost C then there is a non-preemptive schedule with cost at most $2C$. Further, such a non-preemptive schedule can be found in polynomial time.

It would be also interesting to show if the optimal gap between values of preemptive and non-preemptive schedules depends on parameter $\xi = \min_S\{\kappa(S)/|S|\}$. For example, we know if $\xi = 1$ then there is no advantage for preemptive schedules, i.e. $\eta = 1$ in this case.

4 Comparison of our LP and the previous one in [2, 15]

In this section we compare our configuration LP and the LP considered in [2, 15], which is time-indexed and based on knapsack covering inequalities. We show that our configuration LP ($\mathsf{LP}_{\mathsf{Primal}}$) is *stronger* for the non-preemptive problem than the LP ($\mathsf{LP}_{\mathsf{BGK}}$) considered in [2, 15]. We first provide an instance for which $\mathsf{LP}_{\mathsf{Primal}}$ has an objective value strictly larger than $\mathsf{LP}_{\mathsf{BGK}}$. Secondly, we show that for any instance $\mathsf{LP}_{\mathsf{Primal}}$ has an objective no smaller than the objective of $\mathsf{LP}_{\mathsf{BGK}}$. Together with the fact that both LPs are valid relaxations, we establish that $\mathsf{LP}_{\mathsf{Primal}}$ is stronger than $\mathsf{LP}_{\mathsf{BGK}}$. We will include the proofs in the full version of this paper.

References

1. Yossi Azar, Iftah Gamzu, and Xiaoxin Yin. In *STOC*, pages 669–678, 2009.
2. Nikhil Bansal, Anupam Gupta, and Ravishankar Krishnaswamy. A constant factor approximation algorithm for generalized min-sum set cover. In *SODA*, pages 1539–1545, 2010.
3. Nikhil Bansal and Subhash Khot. Optimal long code test with one free bit. In *FOCS*, pages 453–462, 2009.
4. Amotz Bar-Noy, Mihir Bellare, Magnús M. Halldórsson, Hadas Shachnai, and Tami Tamir. On chromatic sums and distributed resource allocation. *Inf. Comput.*, 140(2):183–202, 1998.
5. Amotz Bar-Noy, Magnús M. Halldórsson, and Guy Kortsarz. A matched approximation bound for the sum of a greedy coloring. *Inf. Process. Lett.*, 71(3):135–140, 1999.
6. S. Burer and R. Monteiro. A projected gradient algorithm for solving the maxcut sdp relaxation. *Optimization Methods and Software*, 15:175–200, 2001.
7. Chandra Chekuri and Rajeev Motwani. Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics*, 98(1-2):29–38, 1999.
8. Uriel Feige, László Lovász, and Prasad Tetali. Approximating min sum set cover. *Algorithmica*, 40(4):219–234, 2004.
9. Martin Grotschel, László Laszlo Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization. Second edition. Algorithms and Combinatorics, 2.* Springer-Verlag, Berlin, 1993.
10. Leslie A. Hall, Andreas S. Schulz, David B. Shmoys, and Joel Wein. Scheduling to minimize average completion time: Off-line and on-line approximation algorithms. *Mathematics of Operations Research*, 22(3):513–544, 1997.
11. François Margot, Maurice Queyranne, and Yaoguang Wang. Decompositions, network flows, and a precedence constrained single-machine scheduling problem. *Operations Research*, 51(6):981–992, 2003.
12. Maurice Queyranne and Maxim Sviridenko. A $(2+\varepsilon)$-approximation algorithm for the generalized preemptive open shop problem with minsum objective. *J. Algorithms*, 45(2):202–212, 2002.
13. Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency.* Springer-Verlag, Berlin, 2003.
14. Andreas Schulz and Martin Skutella. Random-based scheduling: new approximations and lp lower bounds. In *RANDOM*, pages 119–133, 1997.
15. Martin Skutella and David P. Williamson. A note on the generalized min-sum set cover problem. *Operations Research Letters*, To appear, 2011.
16. Gerhard J. Woeginger. On the approximability of average completion time scheduling under precedence constraints. *Discrete Applied Mathematics*, 131(1):237–252, 2003.

Randomized Communication Complexity for Linear Algebra Problems over Finite Fields *

Xiaoming Sun[1] and Chengu Wang[2]

1 **Institute of Computing Technology, Chinese Academy of Sciences**
Beijing, China
sunxiaoming@ict.ac.cn

2 **IIIS, Tsinghua University**
Beijing, China
wangchengu@gmail.com

Abstract

Finding the singularity of a matrix is a basic problem in linear algebra. Chu and Schnitger [3] first considered this problem in the communication complexity model, in which Alice holds the first half of the matrix and Bob holds the other half. They proved that the deterministic communication complexity is $\Omega(n^2 \log p)$ for an $n \times n$ matrix over the finite field $\mathbb{F}_p$. Then, Clarkson and Woodruff [4] introduced the singularity problem to the streaming model. They proposed a randomized one pass streaming algorithm that uses $O(k^2 \log n)$ space to decide if the rank of a matrix is k, and proved an $\Omega(k^2)$ lower bound for randomized one-way protocols in the communication complexity model.

We prove that the randomized/quantum communication complexity of the singularity problem over $\mathbb{F}_p$ is $\Omega(n^2 \log p)$, which implies the same space lower bound for randomized streaming algorithms, even for a constant number of passes. The proof uses the framework by Lee and Shraibman [8], but we choose Fourier coefficients as the witness for the dual approximate norm of the communication matrix. Moreover, we use Fourier analysis to show the same randomized/quantum lower bound when deciding if the determinant of a non-singular matrix is a or b for non-zero a and b.

1998 ACM Subject Classification F.2.1 Numerical Algorithms and Problems; G.1.3 Numerical Linear Algebra

Keywords and phrases communication complexity, streaming, matrix, singularity, determinant

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.477

1 Introduction

Communication complexity, introduced by Yao [16], is a powerful tool to solve a variety of problems in areas as disparate as VLSI design, decision trees, data structures and circuit complexity [7]. It is a game between two parties, Alice and Bob, with unlimited computing power, that want to compute the value of a function $f : X \times Y \mapsto \{0, 1\}$, but Alice only knows $x \in X$ while Bob only knows $y \in Y$. The communication complexity is the minimal amount of bits they transfer. We denote the randomized and quantum communication complexity which succeeds with probability at least $1 - \epsilon$ by $R_\epsilon(f)$ and $Q_\epsilon(f)$ (or $Q^*_\epsilon(f)$) respectively, where

* This work was supported in part by the National Basic Research Program of China Grant 2011CBA00300, 2011CBA00301, the National Natural Science Foundation of China Grant 61033001, 61061130540, 61073174. The first author was also supported in part by the National Natural Science Foundation of China Grant 61170062 and Tsinghua University Initiative Scientific Research Program 2009THZ02120.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 477–488

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

$R_\epsilon(f)$ is with private coin, $Q_\epsilon(f)$ is without entanglement and $Q^*_\epsilon(f)$ is with entanglement. The three functions have the following relationship: $Q^*_\epsilon(f) \leq Q_\epsilon(f) \leq O(R_\epsilon(f))$ [6].

In the streaming model, the input is presented as a sequence and can be examined by the algorithm in only a few passes (typically just one). We are interested in the size of memory the algorithm uses. It can be proved by a reduction that the size of memory times the number of passes can be bounded by the communication complexity if Alice holds the first part of the stream and Bob holds the remaining.

We focus on linear algebra problems, because they are fundamental problems in mathematics, and the matrix computation is used everywhere. Furthermore, the singularity problem is the most basic problem, because it can be reduced to many linear algebra problems, e.g. to determine whether linear equations have a solution, to compute the diagonal of the LU decomposition or QR decomposition and to determine whether two subspaces intersect. Formally speaking, for an $n \times n$ matrix x whose entries are in the the finite field $\mathbb{F}_p$, the singularity problem is to decide whether x is singular over $\mathbb{F}_p$, and the determinant problem is to compute the determinant of x over $\mathbb{F}_p$ for non-singular x. In the streaming model, the input x comes row by row sequentially, while in the communication complexity model, Alice holds the first $n/2$ rows, and Bob holds the remaining $n/2$ rows.

These problems have a trivial deterministic algorithm that uses $O(n^2 \log p)$ spaces in one pass or the same number of communications in one-way. Chu and Schnitger [3] proved an $\Omega(n^2 \log p)$ communication complexity for deterministic protocols of the singularity problem. Luo and Tsitsiklis [9] proved that a deterministic protocol must transfer $\Omega(n^2)$ real numbers for the matrix inversion problem. Clarkson and Woodruff [4] proposed a randomized one pass streaming algorithm that uses $O(k^2 \log n)$ space to decide if the rank of a matrix is k, and proved an $\Omega(k^2)$ lower bound for randomized one-way protocol in the communication complexity model by reducing from the index function, which implies an $\Omega(n^2)$ space lower bound in the streaming model with one pass. Deciding the disjointness of two $n/2$ dimensional subspaces is actually the singularity problem. Miltersen et al. [10] showed a tight lower bound when deciding whether a vector is in a subspace of $\mathbb{F}_2^n$ in the one-sided error randomized asymmetric communication complexity model, by using the Richness Lemma.

In the communication model, there is another way to distribute the input: Alice and Bob each holds an $n \times n$ matrix x and y, respectively, and they want to compute the singularity or determinant of $x + y$. The two ways are equivalent up to a constant factor, because

$$\det(x+y) = \det \begin{pmatrix} x+y & \mathbf{0}_{n\times n} \\ y & \mathbf{I}_{n\times n} \end{pmatrix} = \det \begin{pmatrix} x & -\mathbf{I}_{n\times n} \\ y & \mathbf{I}_{n\times n} \end{pmatrix}.$$

This way is more beautiful and symmetric. When $p = 2$, $f(x + y) = f(x \oplus y)$ is a important block-composed function with good properties and attracted lots of attentions recently [11, 18, 14, 17]. Here, we formally define the problems in this way.

▶ **Problem 1** (Singularity). *Alice and Bob hold two $n \times n$ matrices x and y over $\mathbb{F}_p$, separately. They want to determine whether $x + y$ is singular over $\mathbb{F}_p$.*

▶ **Problem 2** ($\text{Det}_{a,b}$). *Alice and Bob hold two $n \times n$ matrices x and y over $\mathbb{F}_p$, separately. For $a, b \in \mathbb{F}_p \setminus \{0\}$, we promise that $\det_p(x+y)$ is either a or b, where $\det_p$ is the determinant over $\mathbb{F}_p$. They want to compute $\det_p(x + y)$.*

1.1 Our Results

▶ **Theorem 3.** *The randomized/quantum communication complexity of* Singularity *is $\Omega(n^2 \log p)$.*

We prove it using the duality of the approximate norm [8]. We compute all the Fourier coefficients of the singularity function and use them as a witness in the Duality Theorem.

This result implies the same lower bound when deciding if two subspaces of $\mathbb{F}_p^n$ intersect. Also, it implies that $\Omega(k^2)$ communication is required to decide if the rank of $x+y$ is k, by padding n^2-k^2 zeros, which improves the determinant result in [4].

▶ **Theorem 4.** *The randomized/quantum communication complexity of* $\textsc{Det}_{a,b}$ *is* $\Omega(n^2 \log p)$, *for all non-zero* a *and* b.

For this problem, we prove a small spectral norm of the matrix representing the problem, by decomposing the matrix onto Fourier basis which has good properties.

All these results imply the same space lower bound for randomized streaming algorithms, even if the algorithm reads the stream a constant number of passes.

1.2 Outline

In Section 2, we define the basic notations. We prove Theorem 3 in Section 3 and Theorem 4 in Section 4. Finally, we discuss the open problem in Section 5.

2 Preliminaries

For prime p, $\mathbb{F}_p$ is a finite field. For a function $f:\mathbb{F}_p^N \mapsto \mathbb{R}$, the Fourier coefficient of f is

$$\hat{f}(s) = \frac{1}{p^N} \sum_{x \in \mathbb{F}_p^N} \omega^{-\langle s,x\rangle} f(x),$$

where $\omega = e^{2\pi i/p}$. The inverse transform is

$$f(x) = \sum_{s \in \mathbb{F}_p^N} \omega^{\langle s,x\rangle} \hat{f}(s).$$

The Kronecker delta, denoted by $\delta_{i,j}$, is 1 if $i=j$ and 0 otherwise.

▶ **Fact 5.** *The number of* $n \times n$ *matrices of rank* r *over* $\mathbb{F}_p$ *is* $p^{r(r-1)/2}\binom{n}{r}_p \prod_{k=n-r+1}^{n}(p^k-1)$. *Especially, the number of non-singular* $n \times n$ *matrices over* $\mathbb{F}_p$ *is* $\prod_{k=0}^{n-1}(p^n-p^k)$.

In this paper, we don't distinguish vectors (matrices) from discrete unary (bivariate) functions. For example, for vector v, $v(x)$ means the x-th element of v, and for matrix A, $A(x,y)$ means the entry at x-th row and y-th column.

For a vector x, we define the ℓ_1-norm $\|x\|_1 = \sum_i |x_i|$, and the ℓ_∞-norm $\|x\|_\infty = \max_i |x_i|$.

For a matrix A, we also define the ℓ_1-norm $\|A\|_1 = \sum_{i,j} |x_{i,j}|$, and the ℓ_∞-norm $\|A\|_\infty = \max_{i,j} |A_{i,j}|$. Let $\sigma = (\sigma_1, \cdots, \sigma_{\mathrm{rank}(A)})$ be the vector of nonzero singular values of A. The trace norm of A is $\|A\|_{\mathrm{tr}} = \|\sigma\|_1$. The spectral norm is $\|A\| = \|\sigma\|_\infty$, which is also the square root of the largest eigenvalue of the positive-semidefinite matrix $A^\dagger A$ [5], where $A^\dagger$ is the conjugate transpose of A.

3 Lower Bound for Singularity Problem

We first introduce the approximate rank and norm. Then, for XOR composed function $g(x \oplus y)$, the trace norm is equal to the ℓ_1 norm of the Fourier coefficients. This property still holds for approximate norm and for $g(x+y)$ in $\mathbb{F}_p$. After that, we present the Duality Theorem, which converts the definition of the approximate norm from min to max. Finally, we compute Fourier coefficients of the singularity function, and choose it as the witness.

3.1 Approximation Norms

The matrix rank and matrix norm can give a lower bound for deterministic communication complexity. Similarly, the approximate rank and norm can prove a lower bound for randomized/quantum protocols.

For $\alpha \geq 1$ and a sign matrix A, we define the approximate trace norm by

$$\|A\|_{\mathrm{tr}}^{\alpha} = \min_{B:1\leq A_{i,j}B_{i,j}\leq\alpha} \|B\|_{\mathrm{tr}},$$

and the approximate rank by

$$\mathrm{rank}^{\alpha}(A) = \min_{B:1\leq A_{i,j}B_{i,j}\leq\alpha} \mathrm{rank}(B).$$

For $\alpha \geq 1$ and a sign vector x, we define the approximate Fourier ℓ_1-norm by

$$\|\hat{x}\|_1^{\alpha} = \min_{y:1\leq x_i y_i\leq\alpha} \|\hat{y}\|_1.$$

▶ **Theorem 6.** *[2] Let A be a sign matrix and $0 < \epsilon < 1/2$, and $\alpha = 1/(1-2\epsilon)$, then $Q_\epsilon(A) \geq \frac{1}{2}\log \mathrm{rank}^{\alpha}(A)$.*

Because the approximate rank can be bounded by the approximate trace norm [8]:

$$\mathrm{rank}^{\alpha}(A) \geq \frac{(\|A\|_{\mathrm{tr}}^{\alpha})^2}{\alpha^2 \cdot \mathrm{size}(A)},$$

the approximate trace norm can give a lower bound for the communication complexity. This result can also be found in [12].

▶ **Lemma 7.** *Let $g : \mathbb{F}_p^N \mapsto \{-1,1\}$ be a sign function, and $f = g\circ+^{\otimes N}$, i.e. $f(x,y) = g(x+y)$. Let A be the sign matrix representing f. Then, $\|A\|_{\mathrm{tr}}^{\alpha} \geq \|\hat{g}\|_1^{\alpha} \cdot p^N$.*

The $p = 2$ case of Lemma 7 can be found in [8, Theorem 85]. It can be easily generalized to $\mathbb{F}_p$ with little changes. As a result, we omit this proof.

Combining them together, the randomized/quantum communication complexity can be bounded by the $\|\hat{\cdot}\|_1^{\alpha}$ norm.

▶ **Corollary 8.** *For $g : \mathbb{F}_p^N \mapsto \{-1,1\}$ and $f = g \circ +^{\otimes N}$, $Q_\epsilon(f) \geq \log \|\hat{g}\|_1^{\alpha} - 2\alpha$, where $\alpha = 1/(1-2\epsilon)$.*

3.2 Duality

The definition of the approximate Fourier ℓ_1 norm begins with $\min_y$. In such a definition, we have to check every y if we want to prove a lower bound. However, the Duality Theorem converts $\min_y$ to $\max_y$. As a result, a particular y, called the witness, is enough to prove a lower bound.

▶ **Definition 9.** For a general norm $\|\cdot\|$ on $\mathbb{R}^N$, the dual norm on $\mathbb{R}^N$, denoted by $\|\cdot\|^*$, is defined by

$$\|x\|^* = \max_{y\in\mathbb{R}^N:\|y\|\leq 1} \langle x, y\rangle.$$

▶ **Theorem 10** (Duality Theorem)**.** *[8, Theorem 64] For a general norm $\|\cdot\|$ on $\mathbb{R}^N$,*

$$\|x\|^{\alpha} = \max_{y:\|y\|^*\leq 1} \frac{1+\alpha}{2}\langle x, y\rangle + \frac{1-\alpha}{2}\|y\|_1.$$

3.3 Choosing Fourier Coefficients as Witness

It is difficult to find a useful witness. The first choice that comes to mind is to choose $h = g$, which can be used to prove the inner product problem. The discrepancy method is the special case of taking $h = \mu \circ g$ for a distribution μ [8]. Here we propose a new choice: taking $h = \hat{g}$. Now we calculate $\hat{g}$ first.

We define a sign function $g : \mathbb{F}_p^{n\times n} \mapsto \{-1, 1\}$, where $g(x) = -1$ if x is full rank over $\mathbb{F}_p$ and $g(x) = 1$ otherwise. Then, we define $f(x, y) = g(x + y)$. In such a definition, f is the function representing the SINGULARITY problem.

We change the value of g from $\{-1, 1\}$ to $\{0, 1\}$, by defining $g_{01} = (1 - g)/2$. In other words, $g_{01}(x) = 1$ if x is full rank, and $g_{01}(x) = 0$ otherwise.

$\widehat{g_{01}}$ has a good property that a same rank results in a same value.

▶ **Lemma 11.** *For $s, t \in \mathbb{F}_p^{n\times n}$, $\widehat{g_{01}}(s) = \widehat{g_{01}}(t)$ if $\mathrm{rank}_p(s) = \mathrm{rank}_p(t)$, where rank_p is the matrix rank over $\mathbb{F}_p$.*

Proof. Because $\mathrm{rank}_p(s) = \mathrm{rank}_p(t)$, there are full rank matrices u and v, such that $s = vtu$. Since v^{T} and u^{T} are full rank, $y = v^{\mathrm{T}}xu^{\mathrm{T}}$ is a bijection between matrices x and y.

$$\widehat{g_{01}}(s) = \frac{\sum_x \omega^{-\mathrm{tr}(s^{\mathrm{T}}x)} g_{01}(x)}{p^{n^2}} = \frac{\sum_x \omega^{-\mathrm{tr}(t^{\mathrm{T}}v^{\mathrm{T}}xu^{\mathrm{T}})} g_{01}(x)}{p^{n^2}} = \frac{\sum_y \omega^{-\mathrm{tr}(t^{\mathrm{T}}y)} g_{01}(y)}{p^{n^2}} = \widehat{g_{01}}(t)$$

◀

▶ **Lemma 12.** *Let $r = \mathrm{rank}_p(s)$, then*

$$\widehat{g_{01}}(s) = (-1)^r p^{-n(n+1)/2} \prod_{k=1}^{n-r} (p^k - 1).$$

Proof. By Lemma 11, we only need to choose one s for each rank to prove it. For rank r, we choose $s = \mathrm{diag}(1, \cdots, 1, 0, \cdots, 0)$, which is a diagonal matrix with r 1's in the diagonal.

We start from the most simple case of $\mathrm{rank}_p(s) = 0$, i.e. s is an all-zero matrix.

$$p^{n^2} \cdot \widehat{g_{01}}(s) = \sum_x \omega^{-\langle \mathbf{0}, x\rangle} g_{01}(x) = \sum_x g_{01}(x) = \#\text{ of full rank matrices} = \prod_{k=0}^{n-1} (p^n - p^k)$$

Then, we consider rank-1 matrix $s = \mathrm{diag}(1, 0, 0, \cdots, 0)$, which is a matrix with all zero entries except for the top left one.

For $k = 0, 1, \cdots, n-1$, we denote the k-th row of matrix x by $x(k, \cdot)$, and the submatrix from the k-th row to the last row by $x(k\text{–}, \cdot)$.

$$p^{n^2} \cdot \widehat{g_{01}}(s) = \sum_{x(0,\cdot)} \sum_{x(1\text{–},\cdot)} \omega^{-\langle s, x\rangle} g_{01}(x) = \sum_{x(0,\cdot)} \omega^{-x(0,0)} \sum_{x(1\text{–},\cdot)} g_{01}(x)$$

$\sum_{x(1\text{–},\cdot)} g_{01}(x)$ is the number of the full rank matrices given the first row. It is $\prod_{k=1}^{n-1}(p^n - p^k)$ if the first row is non-zero, and 0 if the first row is zero. Except for the case that the first row is zero, they are all canceled out because $\sum_{x(0,0)} \omega^{-x(0,0)} = 0$. Thus, the remaining is the minus value of the case that the first row is non-zero, i.e. $p^{n^2} \cdot \widehat{g_{01}}(s) = -\prod_{k=1}^{n-1}(p^n - p^k)$.

In general, for the rank-r matrix $s = \mathrm{diag}(1, \cdots, 1, 0, \cdots, 0)$ with r 1's in the diagonal,

$$p^{n^2} \cdot \widehat{g_{01}}(s) = \sum_{x(0,\cdot)} \omega^{-x(0,0)} \sum_{x(1,\cdot)} \omega^{-x(1,1)} \cdots \sum_{x(r-1,\cdot)} \omega^{-x(r-1,r-1)} \sum_{x(r\text{–},\cdot)} g_{01}(x).$$

$\sum_{x(r-,\cdot)} g_{01}(x)$ is the number of full rank matrices given the first r rows. It is $\prod_{k=r}^{n-1}(p^n - p^k)$ if the first r rows is linear independent, and 0 otherwise.

We define ζ as follows: $\zeta(x,k) = 0$ if the first k rows of x are linear dependent and $\zeta(x,k) = 1$ otherwise. Now we have

$$p^{n^2} \cdot \widehat{g_{01}}(s) = \sum_{x(0,\cdot)} \omega^{-x(0,0)} \sum_{x(1,\cdot)} \omega^{-x(1,1)} \cdots \sum_{x(r-1,\cdot)} \omega^{-x(r-1,r-1)} \zeta(x,r) \prod_{k=r}^{n-1} (p^n - p^k)$$

Then we slightly change ζ to $\bar{\zeta}$: $\bar{\zeta}(x,k) = 0$ if the first $k-1$ rows of x are linear independent but the first k rows are linear dependent, and $\bar{\zeta}(x,k) = 1$ otherwise. It is clear that $\zeta(x,r) = \prod_{k=1}^{r} \bar{\zeta}(x,k)$, which gives

$$p^{n^2} \cdot \widehat{g_{01}}(s) = \sum_{x(0,\cdot)} \omega^{-x(0,0)} \bar{\zeta}(x,1) \sum_{x(1,\cdot)} \omega^{-x(1,1)} \bar{\zeta}(x,2) \cdots \sum_{x(r-1,\cdot)} \omega^{-x(r-1,r-1)} \bar{\zeta}(x,r) \prod_{k=r}^{n-1} (p^n - p^k).$$

Since $\sum_{x(k,\cdot)} \omega^{-x(k,k)} = 0$, we have

$$\sum_{x(k,\cdot)} \omega^{-x(k,k)} \bar{\zeta}(x,k+1) = \sum_{x(k,\cdot):\bar{\zeta}(x,k+1)=1} \omega^{-x(k,k)} = -\sum_{x(k,\cdot):\bar{\zeta}(x,k+1)=0} \omega^{-x(k,k)}.$$

$x(k,\cdot)$ goes over the linear combinations of the first k rows. At the same time, $x(k,k)$ goes over the linear combinations of $x(0,k), x(1,k), \cdots, x(k-1,k)$. If $x(0,k), x(1,k), \cdots, x(k-1,k)$ are not all zero, $x(k,k)$ is balanced, so $-\sum \omega^{-x(k,k)} = 0$. If $x(0,k), x(1,k), \cdots, x(k-1,k)$ are all zero, $x(k,k) = 0$, so $-\sum \omega^{-x(k,k)} = -p^k$.

Consequently, we have

$$p^{n^2} \cdot \widehat{g_{01}}(s) = (-1)(-p)(-p^2)\cdots(-p^{r-1}) \prod_{k=r}^{n-1} (p^n - p^k) = (-1)^r p^{-n(n+1)/2} \prod_{k=1}^{n-r} (p^k - 1).$$

◀

▶ **Lemma 13.** $\|\hat{g}(s)\|_1 < 1 + 6 \cdot p^{-n} \prod_{k=1}^{n} (p^k - 1)$

Proof.

$$\begin{aligned}
\|\hat{g}(s)\|_1 &\leq 1 + 2\|\widehat{g_{01}}(s)\|_1 \\
&= 1 + 2\sum_{r=0}^{n} \sum_{s:\mathrm{rank}_p(s)=r} |\widehat{g_{01}}(s)| \\
&= 1 + 2\sum_{r=0}^{n} p^{r(r-1)/2} \binom{n}{r}_p \prod_{k=n-r+1}^{n} (p^k - 1) \cdot p^{-n(n+1)/2} \prod_{k=1}^{n-r} (p^k - 1) \\
&= 1 + 2p^{-n(n+1)/2} \prod_{k=1}^{n} (p^k - 1) \sum_{r=0}^{n} p^{r(r-1)/2} \binom{n}{r}_p \\
&= 1 + 2p^{-n(n+1)/2} \prod_{k=1}^{n} (p^k - 1) \prod_{k=0}^{n-1} (1 + p^k) \\
&= 1 + 2p^{-n} \prod_{k=1}^{n} (p^k - 1) \prod_{k=0}^{n-1} \frac{1 + p^k}{p^k} \\
&< 1 + 2p^{-n} \prod_{k=1}^{n} (p^k - 1) \cdot 3
\end{aligned}$$

◀

Now we can prove that the approximate Fourier ℓ_1-norm of g is large.

▶ **Lemma 14.**

$$\|\hat{g}\|_1^{3/2} = p^{\Omega(n^2)}$$

Proof. In the proof of Lemma 7, we know $\|\hat{h}\|_1^* = p^{n^2}\|\hat{h}\|_\infty$. We rewrite Theorem 10:

$$\|\hat{g}\|_1^\alpha = \max_{h:p^{n^2}\|\hat{h}\|_\infty \le 1} \frac{1+\alpha}{2}\langle g, h\rangle + \frac{1-\alpha}{2}\|h\|_1.$$

We choose $h = (-1)^{n+1}\widehat{g_{01}}$. So, $\hat{h} = (-1)^{n+1}g_{01}/p^{n^2}$, and $\|\hat{h}\|_\infty = 1/p^{n^2}$.

$$\begin{aligned}
\langle g_{01}, h\rangle &= \langle g_{01}, (-1)^{n+1}\widehat{g_{01}}\rangle \\
&= (-1)^{n+1}\sum_s g_{01}(s)\widehat{g_{01}}(s) \\
&= (-1)^{n+1}\sum_{s:\mathrm{rank}_p(s)=n} \widehat{g_{01}}(s) \\
&= (-1)^{n+1}\prod_{k=0}^{n-1}(p^n - p^k)\cdot(-1)^n p^{-n(n+1)/2} \\
&= -p^{-n}\prod_{k=1}^{n}(p^k - 1)
\end{aligned}$$

$$\langle g, h\rangle = -2\langle g_{01}, h\rangle + \sum_x h(x) = -2\cdot -p^{-n}\prod_{k=1}^{n}(p^k-1) + 0 = 2p^{-n}\prod_{k=1}^{n}(p^k-1)$$

$$\begin{aligned}
\|\hat{g}\|_1^{3/2} &\ge \frac{1+3/2}{2}\langle g, h\rangle + \frac{1-3/2}{2}\|h\|_1 \\
&\ge \frac{5}{4}\cdot 2p^{-n}\prod_{k=1}^{n}(p^k-1) - \frac{1}{4}\left(6p^{-n}\prod_{k=1}^{n}(p^k-1) + 1\right) \\
&= p^{-n}\prod_{k=1}^{n}(p^k-1) - \frac{1}{4} \\
&\ge p^{-n}\prod_{k=1}^{n}p^{k-1} - \frac{1}{4} \\
&= p^{n(n-3)/2} - \frac{1}{4}
\end{aligned}$$

◀

▶ **Theorem 15** (Theorem 3 Restated). *The randomized/quantum communication complexity of the* Singularity *problem is* $\Omega(n^2\log p)$.

Proof. By Corollary 8, $Q_{1/6}(f) \ge \log\|\hat{g}\|_1^{3/2} - 3 = \Omega(n^2\log p)$. $R_{1/6}(f) = \Omega(Q_{1/6}(f))$. ◀

4 Lower Bound for Determinant of Non-singular Matrix

We use a small spectral norm of the matrix representing the $\mathrm{Det}_{a,b}$ problem to prove the communication complexity lower bound. To prove the lower bound of spectral norm, we decompose the matrix onto Fourier basis, because the Fourier basis has good properties and a small spectral norm.

4.1 Spectral Norm Method

In the previous section, a large trace norm implies a large communication complexity lower bound. However, here we use a small spectral norm to prove a large communication complexity lower bound.

The spectral norm method is based on the discrepancy method, which can derive the communication complexity lower bound by giving an upper bound for a value called discrepancy defined below.

▶ **Definition 16** (Discrepancy). Let $f : \mathbb{F}_p^N \times \mathbb{F}_p^N \mapsto \{0,1\}$ be a function, $S \times T$ be a rectangle, and μ be a probability distribution on $\mathbb{F}_p^N \times \mathbb{F}_p^N$. Denote $\mathrm{disc}_\mu(S \times T, f) = |\sum_{(x,y)\in S\times T} \mu(x,y)(-1)^{f(x,y)}|$, and $\mathrm{disc}_\mu(f) = \max_{S,T\subseteq\mathbb{F}_p^N} \mathrm{disc}_\mu(S \times T, f)$.

The discrepancy is widely used in proving communication complexity lower bound [1, 15, 7], with many applications. It was also used to prove the quantum lower bound [6, 13], and could be phrased in the following theorem.

▶ **Theorem 17.** *[6] For any function f and any distribution μ, we have*

$$Q_\epsilon^*(f) = \Omega\left(\log \frac{1-2\epsilon}{\mathrm{disc}_\mu(f)}\right).$$

Furthermore, the discrepancy can be bounded by the spectral norm.

▶ **Theorem 18.** *[7, Example 3.29] Let $f : \mathbb{F}_p^N \times \mathbb{F}_p^N \mapsto \{0,1,\perp\}$ be a partial Boolean function. We define the corresponding partial sign matrix F by its entries*

$$F(x,y) = \begin{cases} 1 & \text{if } f(x,y) = 0, \\ -1 & \text{if } f(x,y) = 1, \\ 0 & \text{otherwise.} \end{cases}$$

For the uniform distribution μ on the defined inputs of f, we have $\mathrm{disc}_\mu(f) \le p^N \cdot \|F\|/\|F\|_1$.

4.2 Fourier Basis Matrix

For the determinant problem, it is difficult to compute $\|F\|$ directly. We will decompose F into Fourier basis: $F = \sum_k \lambda_k H_k$. The spectral norm of the Fourier basis H_k is easier to compute. At last, we will use the triangle inequality to bound $\|F\|$.

▶ **Definition 19** (Discrete logarithm). $\mathbb{F}_p^* = \mathbb{F}_p \setminus \{0\}$ is a cyclic multiplicative group, in which 2 is a primitive element (the generator for the multiplicative group). For $a \in \mathbb{F}_p^*$, we say $k = \log_2 a$ if $2^k = a$.

Let $\eta = e^{2\pi i/(p-1)}$. For $k = 0, 1, \cdots, p-2$ and $a \in \mathbb{F}_p$, we define a family of τ as below.

$$\tau_k(a) = \begin{cases} 0 & \text{if } a = 0, \\ \eta^{k \log_2 a} & \text{otherwise.} \end{cases}$$

▶ **Lemma 20.** *For $a, b \in \mathbb{F}_p$,*

1. $\tau_k(a)\tau_k(b) = \tau_k(ab)$;
2. $\sum_{a=0}^{p-1} \tau_k(a) = 0$, *if* $k \neq 0$;
3. $\tau_k(a)\tau_k(a)^*\tau_k(a) = \tau_k(a)$.

Proof. 1. If $a = 0$ or $b = 0$, both sides are 0. If $a \neq 0$ and $b \neq 0$, $\tau_k(a)\tau_k(b) = \eta^{k \log_2 a}\eta^{k \log_2 b} = \eta^{k \log_2 (ab)} = \tau_k(ab)$.

2.
$$\sum_{a=0}^{p-1} \tau_k(a) = \sum_{a=1}^{p-1} \tau_k(a) = \sum_{a=1}^{p-1} \eta^{k \log_2 a} = \sum_{l=0}^{p-2} \eta^{kl} = 0$$

3. If $a = 0$, both sides are 0. If $a \neq 0$, $\tau_k(a)^*\tau_k(a) = |\tau_k(a)|^2 = 1$. ◀

Then, we define a family of h_k. For $x \in \mathbb{F}_p^{n\times n}$, we define $h_k(x) = \tau_k(\det_p(x))$.

▶ **Lemma 21.** *For $x, y \in \mathbb{F}_p^{n\times n}$,*
1. *$h_k(x)h_k(y) = h_k(xy)$;*
2. *$h_k(x)h_k(x)^*h_k(x) = h_k(x)$.*

Proof.
1. $h_k(x)h_k(y) = \tau_k(\det_p(x))\tau_k(\det_p(y)) = \tau_k(\det_p(x)\det_p(y)) = \tau_k(\det_p(xy)) = h_k(xy)$;
2. $h_k(x)h_k(x)^*h_k(x) = \tau_k(\det_p(x))\tau_k(\det_p(x))^*\tau_k(\det_p(x)) = \tau_k(\det_p(x)) = h_k(x)$. ◀

▶ **Lemma 22.** *For $t \in \mathbb{F}_p^{n\times n}$, $\hat{h_k}(t) = 0$ if the first row of t is all-zero and $k \neq 0$.*

Proof.
$$\hat{h_k}(t) = \sum_x h(x)\omega^{\langle x,t\rangle} = \sum_{x(1-,\cdot)} \omega^{\langle x(1-,\cdot),t(1-,\cdot)\rangle} \sum_{x(0,\cdot)} h(x)$$

If $x(1-,\cdot)$ is fixed and $x(0,\cdot)$ goes over $\mathbb{F}_p^n$, $\det_p(x)$ is balanced on all non-zero values. Thus, $\sum_{x(0,\cdot)} h(x) = 0$. ◀

▶ **Lemma 23.** *For $w, t \in \mathbb{F}_p^{n\times n}$, if $\det(w) \neq 0$, then $\hat{h_k}(wt) = h_k(w)^*\hat{h_k}(t)$.*

Proof.
$$\begin{aligned}
\hat{h_k}(wt) &= h_k(w)^*h_k(w)\hat{h_k}(wt) \\
&= h_k(w)^*h_k(w)\frac{1}{p^{n^2}}\sum_x h_k(x)\omega^{-\langle x,wt\rangle} \\
&= h_k(w)^*h_k(w^{\mathrm{T}})\frac{1}{p^{n^2}}\sum_x h_k(x)\omega^{-\langle w^{\mathrm{T}}x,t\rangle} \\
&= h_k(w)^*\frac{1}{p^{n^2}}\sum_x h_k(w^{\mathrm{T}}x)\omega^{-\langle w^{\mathrm{T}}x,t\rangle} \\
&= h_k(w)^*\hat{h_k}(t)
\end{aligned}$$

◀

▶ **Lemma 24.** *For $t \in \mathbb{F}_p^{n\times n}$, if $\det(t) = 0$ and $k \neq 0$, then $\hat{h_k}(t) = 0$.*

Proof. Because t is singular, we can find an invertible matrix w such that the first row of wt is all zero. By Lemma 22 and Lemma 23, $\hat{h_k}(t) = \hat{h_k}(wt)/h_k(w)^* = 0/h_k(w)^* = 0$. ◀

▶ **Lemma 25.** *For $k \neq 0$ and $t \in \mathbb{F}_p^{n\times n}$, $\hat{h_k}(t) = \hat{h_k}(I)h_k(t)^*$, where I is the identity matrix of size $n \times n$.*

Proof. If $\det(t) = 0$, $\hat{h_k}(t) = 0 = \hat{h_k}(I) \cdot 0 = \hat{h_k}(I)h_k(t)^*$.
If $\det(t) \neq 0$, $\hat{h_k}(t) = \hat{h_k}(t \cdot I) = h_k(t)^*\hat{h_k}(I)$. ◀

▶ **Lemma 26.** *For $k \neq 0$, $\hat{h_k}(I)^*\hat{h_k}(I) = p^{-n^2}$.*

Proof.

$$\begin{aligned}
\langle \hat{h_k}, \hat{h_k} \rangle &= \sum_x \hat{h_k}(x)^* \hat{h_k}(x) \\
&= \sum_x \left(\hat{h_k}(I) h_k(x) \right)^* \left(\hat{h_k}(I) h_k(x) \right) \\
&= \hat{h_k}(I)^* \hat{h_k}(I) \sum_x h_k(x)^* h_k(x) \\
&= \hat{h_k}(I)^* \hat{h_k}(I) \langle h_k, h_k \rangle \\
&= \hat{h_k}(I)^* \hat{h_k}(I) \cdot p^{n^2} \langle \hat{h_k}, \hat{h_k} \rangle
\end{aligned}$$

Therefore, $\hat{h_k}(I)^* \hat{h_k}(I) = p^{-n^2}$. ◀

At last, we define a family of matrix H_k. For $x, y \in \mathbb{F}_p^{n \times n}$, we define $H_k(x, y) = h_k(x + y)$.

▶ **Lemma 27.** *For $k \neq 0$, $H_k H_k^\dagger H_k = p^{n^2} \cdot H_k$.*

Proof. For $w, x, y, z, r, s, t \in \mathbb{F}_p^{n \times n}$,

$$\begin{aligned}
& (H_k H_k^\dagger H_k)(w, z) \\
= & \sum_x \sum_y H_k(w, x) H_k(y, x)^* H_k(y, z) \\
= & \sum_x \sum_y h_k(w + x) h_k(x + y)^* h_k(y + z) \\
= & \sum_x \sum_y \left(\sum_r \hat{h_k}(r) \omega^{\langle r, w+x \rangle} \right) \left(\sum_s \hat{h_k}(s) \omega^{\langle s, x+y \rangle} \right)^* \left(\sum_t \hat{h_k}(t) \omega^{\langle t, y+z \rangle} \right) \\
= & \sum_r \sum_s \sum_t \hat{h_k}(r) \hat{h_k}(s)^* \hat{h_k}(t) \omega^{\langle r, w \rangle} \left(\sum_x \omega^{\langle r-s, x \rangle} \right) \left(\sum_y \omega^{\langle -s+t, y \rangle} \right) \omega^{\langle t, z \rangle} \\
= & \sum_r \sum_s \sum_t \hat{h_k}(r) \hat{h_k}(s)^* \hat{h_k}(t) \omega^{\langle r, w \rangle} \left(p^{n^2} \delta_{r,s} \right) \left(p^{n^2} \delta_{s,t} \right) \omega^{\langle t, z \rangle} \\
= & \; p^{2n^2} \sum_r \hat{h_k}(r) \hat{h_k}(r)^* \hat{h_k}(r) \omega^{\langle r, w \rangle} \omega^{\langle r, z \rangle} \\
= & \; p^{2n^2} \sum_r (\hat{h_k}(I) h_k(r)^*)(\hat{h_k}(I)^* h_k(r))(\hat{h_k}(I) h_k(r)^*) \omega^{\langle r, w+z \rangle} \\
= & \; p^{2n^2} \sum_r \hat{h_k}(I) \hat{h_k}(I)^* \hat{h_k}(I) h_k(r)^* h_k(r) h_k(r)^* \omega^{\langle r, w+z \rangle} \\
= & \; p^{2n^2} \sum_r p^{-n^2} \hat{h_k}(I) h_k(r)^* \omega^{\langle r, w+z \rangle} \\
= & \; p^{n^2} \sum_r \hat{h_k}(r) \omega^{\langle r, w+z \rangle} \\
= & \; p^{n^2} h_k(w + z) \\
= & \; p^{n^2} H_k(w, z)
\end{aligned}$$

◀

▶ **Lemma 28.** *For $k \neq 0$,*

$$\|H_k\| = p^{n^2/2}.$$

Proof. We denote the largest eigenvalue of a semi-definite matrix A by $\max\operatorname{eval}(A)$. Thus, $\|A\| = \sqrt{\max\operatorname{eval}(A^\dagger A)}$.

$$\begin{aligned}\|H_k H_k^\dagger H_k\| &= \sqrt{\max\operatorname{eval}((H_k H_k^\dagger H_k)^\dagger \cdot H_k H_k^\dagger H_k)} \\ &= \sqrt{\max\operatorname{eval}((H_k^\dagger H_k)^3)} \\ &= \left(\sqrt{\max\operatorname{eval}(H_k^\dagger H_k)}\right)^3 \\ &= \|H_k\|^3\end{aligned}$$

Comparing to Lemma 27, we have $\|H_k\|^3 = p^{n^2}\|H_k\|$. Therefore, $\|H_k\| = p^{n^2/2}$. ◀

4.3 Lower Bound for Det

For $\textsc{Det}_{a,b}$, $f(x,y)$ is 0 if $\det_p(x+y) = a$, 1 if $\det_p(x+y) = b$ and undefined for other cases. We define the corresponding partial sign matrix F as below.

$$F(x,y) = \begin{cases} 1 & \text{if } \det_p(x+y) = a, \\ -1 & \text{if } \det_p(x+y) = b, \\ 0 & \text{otherwise.} \end{cases}$$

▶ **Lemma 29.** $\|F\|_1 = \Omega(p^{2n^2-1})$.

Proof. In the uniform distribution, $\Pr_{x\in\mathbb{F}_p^{n\times n}}[\det_p(x) \neq 0] = \prod_{k=1}^{n}(1-p^{-k}) > \phi(p^{-1}) > 0$, where ϕ is the Euler function. This means that the density of the non-singular matrix is greater than a constant. Furthermore, the determinant is balanced on all non-zero values. Thus, $\Pr_{x\in\mathbb{F}_p^{n\times n}}[\det_p(x) = a \text{ or } b] > \frac{2}{p-1}\phi(p^{-1})$. Finally, $\|F\|_1 \geq \frac{2}{p-1}\phi(p^{-1})\cdot p^{2n^2} = \Omega(p^{2n^2-1})$. ◀

▶ **Lemma 30.** $\|F\| \leq 2p^{n^2/2}$.

Proof. It is easy to check that we can decompose F to the Fourier basis matrices:

$$F = \frac{1}{p}\cdot\sum_{k=1}^{p-2}\left(\eta^{-k\log_2 a} - \eta^{-k\log_2 b}\right)H_k$$

. Then, we use the triangle inequality of matrix norm to bound the norm of F:

$$\|F\| \leq \frac{1}{p}\cdot\sum_{k=1}^{p-2}\left|\eta^{-k\log_2 a} - \eta^{-k\log_2 b}\right|\|H_k\| \leq \frac{1}{p}\cdot(p-2)\cdot 2\cdot p^{n^2/2} < 2p^{n^2/2}.$$

◀

▶ **Theorem 31** (Theorem 4 Restated)**.** *The randomized/quantum communication complexity of* $\textsc{Det}_{a,b}$ *is* $\Omega(n^2\log p)$.

Proof.

$$\operatorname{disc}_\mu(f) \leq p^{n^2}\frac{\|F\|}{\|F\|_1} \leq p^{n^2}\cdot\frac{2p^{n^2/2}}{\Omega(p^{2n^2-1})} = O(p^{-n^2/2+1})$$

$$R_{1/3}(f) = \Omega(Q^*_{1/3}(f)),\ Q^*_{1/3}(f) = \Omega\left(\log\frac{1}{\operatorname{disc}_\mu(f)}\right) = \Omega(n^2\log p).$$

◀

5 Open Problems

One open problem is to distinguish between $\det_p(x) = 0$ and $\det_p(x) = a$. We guess it has an $\Omega(n^2 \log p)$ lower bound even for quantum protocols. The proof could be similar to Section 3.

The other one is to compute the (i, j)-th element of the inverse of matrix x. We conjecture the quantum communication $\Omega(n^2 \log p)$ as well. Actually, this problem is as hard as solving linear equations.

We discuss all these problems over $\mathbb{F}_p$, but we think they are still hard over integers.

References

1 László Babai, Peter Frankl, and Janos Simon. Complexity classes in communication complexity theory. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, pages 337–347, Washington, DC, USA, 1986. IEEE Computer Society.

2 Harry Buhrman and Ronald de Wolf. Communication complexity lower bounds by polynomials. In *IEEE Conference on Computational Complexity*, pages 120–130, 2001.

3 J. I. Chu and G. Schnitger. Communication complexity of matrix computation over finite fields. *Theory of Computing Systems*, 28:215–228, 1995. 10.1007/BF01303056.

4 Kenneth L. Clarkson and David P. Woodruff. Numerical linear algebra in the streaming model. In *Proceedings of the 41st annual ACM symposium on Theory of computing*, STOC '09, pages 205–214, New York, NY, USA, 2009. ACM.

5 Roger A. Horn and Charles R. Johnson. *Matrix analysis*. Cambridge Univ Pr, 1990.

6 I. Kremer. *Quantum communication*. PhD thesis, Citeseer, 1995.

7 E. Kushilevitz and N. Nisan. *Communication Complexity*. Cambridge Univ Pr, 1997.

8 Troy Lee and Adi Shraibman. Lower bounds in communication complexity: A survey.

9 Zhi-Quan Luo and John N. Tsitsiklis. On the communication complexity of distributed algebraic computation. *J. ACM*, 40:1019–1047, November 1993.

10 Peter Bro Miltersen, Noam Nisan, Shmuel Safra, and Avi Wigderson. On data structures and asymmetric communication complexity. In *Proceedings of the 27th annual ACM symposium on Theory of computing*, STOC '95, pages 103–111, New York, NY, USA, 1995. ACM.

11 Ran Raz. Fourier analysis for probabilistic communication complexity. *Computational Complexity*, 5:205–221, 1995. 10.1007/BF01206318.

12 AA Razborov. Quantum communication complexity of symmetric predicates. *Izvestiya: Mathematics*, 67:145–159, 2003.

13 Alexander A. Sherstov. The pattern matrix method for lower bounds on quantum communication. In *Proceedings of the 40th annual ACM symposium on Theory of computing*, STOC '08, pages 85–94, New York, NY, USA, 2008. ACM.

14 Yaoyun Shi and Yufan Zhu. Quantum communication complexity of block-composed functions. *Quantum Information and Computation*, 9:444–460, May 2009.

15 Andrew C. Yao. Lower bounds by probabilistic arguments. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, pages 420–428, Washington, DC, USA, 1983. IEEE Computer Society.

16 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing(preliminary report). In *Proceedings of the 7th annual ACM symposium on Theory of computing*, STOC '79, pages 209–213, New York, NY, USA, 1979. ACM.

17 Zhiqiang Zhang and Yaoyun Shi. Communication complexities of symmetric xor functions. *Quantum Information and Computation*, 9:255–263, March 2009.

18 Zhiqiang Zhang and Yaoyun Shi. On the parity complexity measures of boolean functions. *Theoretical Computer Science*, 411(26-28):2612–2618, 2010.

Regular tree languages, cardinality predicates, and addition-invariant FO

Frederik Harwath and Nicole Schweikardt

Institut für Informatik
Goethe-Universität Frankfurt am Main, Germany
Email: {harwath,schweika}@cs.uni-frankfurt.de
URL: http://www.tks.cs.uni-frankfurt.de/{harwath,schweika}

Abstract

This paper considers the logic $\mathrm{FO}_{\mathrm{card}}$, i.e., first-order logic with *cardinality predicates* that can specify the size of a structure modulo some number. We study the expressive power of $\mathrm{FO}_{\mathrm{card}}$ on the class of languages of ranked, finite, labelled trees with successor relations.

Our first main result characterises the class of $\mathrm{FO}_{\mathrm{card}}$-definable tree languages in terms of algebraic closure properties of the tree languages. As it can be effectively checked whether the language of a given tree automaton satisfies these closure properties, we obtain a decidable characterisation of the class of regular tree languages definable in $\mathrm{FO}_{\mathrm{card}}$.

Our second main result considers first-order logic with unary relations, successor relations, and two additional designated symbols $<$ and $+$ that must be interpreted as a linear order and its associated addition. Such a formula is called *addition-invariant* if, for each fixed interpretation of the unary relations and successor relations, its result is independent of the particular interpretation of $<$ and $+$. We show that the $\mathrm{FO}_{\mathrm{card}}$-definable tree languages are exactly the regular tree languages definable in addition-invariant first-order logic.

Our proof techniques involve tools from algebraic automata theory, reasoning with locality arguments, and the use of logical interpretations. We combine and extend methods developed by Benedikt and Segoufin (ACM ToCL, 2009) and Schweikardt and Segoufin (LICS, 2010).

1998 ACM Subject Classification F.4.1 Mathematical Logic, F.4.3 Formal Languages

Keywords and phrases regular tree languages, algebraic closure properties, decidable characterisations, addition-invariant first-order logic, logical interpretations

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.489

1 Introduction

The search for decidable characterisations of certain classes of languages has a long tradition in logic and automata theory. For the case of *word languages* definable by first-order logic FO and extensions thereof, the situation is quite well-understood by now. For example, the languages definable by FO over linearly ordered word structures are exactly the *aperiodic* languages [10], and the languages definable by FO on word structures with successor relation (but without order) are precisely the aperiodic languages *closed under idempotent-guarded swaps* [1]. Similar results are known for extensions of FO such as, e.g., the logics $\mathrm{FO}_{\mathrm{mod}}$ and $\mathrm{FO}_{\mathrm{card}}$ that enrich FO by quantifiers that count modulo some integer, respectively, by predicates that specify the size of the word modulo some integer [11, 9]. All these characterisations lead to effective procedures for deciding whether a given regular language is definable by the respective logic. We refer to [11] for a detailed overview.

Transferring such characterisations from word languages to *tree* languages is usually quite a challenge. In particular, it is a longstanding open problem to find a decidable characterisation

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 489–500

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


of the regular tree languages definable by FO on tree structures with prefix-order (i.e., the transitive closure of the parent-child relation). For trees with successor relations (and without prefix-order), according results for FO and $\mathrm{FO}_{\mathrm{mod}}$ have been achieved in [2], using a new notion of *closure under guarded swaps*.

The present paper transfers techniques of [2] from FO to $\mathrm{FO}_{\mathrm{card}}$, generalising results of [9] from word languages to tree languages. We consider languages of ranked, finite, labelled trees with successor relations (and without prefix-order) definable in $\mathrm{FO}_{\mathrm{card}}$. Our first main result identifies a new property of tree languages called *closure under transfer* and shows that the $\mathrm{FO}_{\mathrm{card}}$-definable tree languages coincide with the regular tree languages that are closed under transfer and under guarded swaps. This leads to a decidable characterisation of the $\mathrm{FO}_{\mathrm{card}}$-definable regular tree languages.

Our second main result considers first-order logic with unary relations, successor relations, and two additional designated symbols $<$ and $+$ that must be interpreted as a linear order and its associated addition. Such a formula is called *addition-invariant* if, for each fixed interpretation of the unary relations and successor relations, its result is independent of the particular interpretation of $<$ and $+$. For some background on addition-invariant first-order logic we refer to [7, 9]. The present paper's second main result shows that the $\mathrm{FO}_{\mathrm{card}}$-definable tree languages are exactly the regular tree languages definable in addition-invariant first-order logic. Our proof techniques involve tools from algebraic automata theory [11, 2], reasoning with locality arguments [7, 6], and the use of logical interpretations (cf., e.g., [7, 5]). In particular, we combine and extend methods developed in [2, 9].

Structure of the paper. We start by fixing the necessary notations in section 2. In section 3, we state and prove our algebraic characterisation of the $\mathrm{FO}_{\mathrm{card}}$-definable tree languages. Section 4 shows that it can be effectively decided whether a given regular tree language has the closure properties associated with $\mathrm{FO}_{\mathrm{card}}$-definability. In section 5, we consider addition-invariant FO and show that the $\mathrm{FO}_{\mathrm{card}}$-definable tree languages are exactly the regular tree languages definable in addition-invariant FO.

Due to space limitations, many technical details of our proofs are deferred to the full version of this paper, available at the authors' websites.

2 Preliminaries

We write $\mathbb{N}$ for the set of natural numbers starting with 0, and $\mathbb{N}_{\geq 1}$ for $\mathbb{N} \setminus \{0\}$. The notation $[n, m]$ is used for the closed interval of natural numbers between n and m. We use the abbreviations $[n] := [0, n]$, $(n] := [1, n]$ and $[n) := [0, n-1]$. By $x \bmod m$ we denote the non-negative remainder when dividing x by m.

We consider *tree languages* of finite trees that are labelled with symbols of a finite alphabet Σ (fixed for the course of the paper). We assume that each node has at most two children, called left child and right child, respectively. This is done for the ease of exposition; all our results easily generalise to arbitrary ranked finite labelled trees. Words are identified with trees where every node has at most one child. We write $\lhd$ (resp. $\trianglelefteq$) for the transitive (resp. reflexive-transitive) closure of the parent-child relation. On each tree, there exists a canonical linear order of the nodes of the tree according to the order in which they are visited by a breadth-first-traversal, where the left child of a node is visited before the right child. We refer to this ordering as the *bf-order* of a tree. The *size* of a tree t, denoted $|t|$, is the number of nodes of t.

A tree is identified with a logical structure, whose universe consists of all nodes of the

tree. For each $a \in \Sigma$, it contains unary relations P_a for the set of nodes with label a, and binary relations S_1 resp. S_2 for the left resp. right child relation. The set of all formulae of first-order logic with these relation symbols is denoted by FO.

We now introduce some basic concepts that will be used throughout this article to talk about the shape of trees. Let t be a tree. For a node v of t, we denote the *subtree rooted at* v by $t_{|v}$. The *k-spill* of v in t, denoted by $t^k_{|v}$, is the restriction of $t_{|v}$ to all vertices with distance at most k from v. The equivalence class of $t^k_{|v}$ under isomorphism is called the *k-type* of v in t. We say that v *realises* its k-type in t. Two nodes (in, potentially, distinct trees) are k-similar, if they realise the same k-type. Two trees are k-similar if their roots are k-similar. For each k-type τ, $|t|_\tau$ is the number of nodes of t that realise τ. If $|t|_\tau > 0$, then τ *occurs* in t. For a tree s, we write $s \leq_k t$ if $|s|_\tau \leq |t|_\tau$ holds for all k-types τ. We use $s =_k t$ and $s <_k t$ analogously. These notations are extended to finite sequences $(t_i)_{i\in(n]}$ of trees by the definition $|(t_i)_{i\in(n]}|_\tau = |t_1|_\tau + \cdots + |t_n|_\tau$.

An *n-context* C, for $n \in \mathbb{N}_{\geq 1}$, is a tree with distinguished leaves $h_1, \ldots, h_n$, called *holes*. If $n = 1$, C is plainly called a *context*. The *inner tree* of C is the tree obtained from C by removing its holes. The size $|C|$ of C is the size of its inner tree. By replacing a hole h_i of C by a tree t (resp. context) one obtains an $(n-1)$-context (resp. n-context). Given trees $t_1, \ldots, t_n$, let $C[t_1, \ldots, t_n]$ be the tree obtained from C by replacing the hole h_i by t_i, for all $i \in (n]$. For a context C and a tree t, $Ct := C \cdot t := C[t]$ is the *concatenation* of C with t. We mostly use contexts as means to decompose given trees. For a tree t with a node u and nodes $v_1, \ldots, v_n$ below u, let $t[u, v_1, \ldots, v_n)$ be the n-context obtained from $t_{|u}$ by removing all nodes strictly below $v_1, \ldots, v_n$ and making $v_1, \ldots, v_n$ holes. Usually, the k-type of a hole's parent in this n-context will not equal its k-type in t. For this reason, we introduce the following concepts.

Let C be a context with a hole h. A k-type-labelling of C is a labelling λ of the nodes of C that assigns $(k+1)$-types to the nodes of the inner tree of C, and a k-type to h. A tree t is *compatible* with λ, if the $(k+1)$-type of t induces $\lambda(h)$. If there exists such a tree t and $\lambda(v)$ is the $(k+1)$-type of v in $C \cdot t$, for each $v \in C$ with $v \neq h$, then λ is *consistent*. A context C together with a *consistent* k-type-labelling λ of C is called a *k-abstract context*. All concepts introduced for trees will be used for abstract contexts as well. When we refer to the types of nodes in abstract contexts, we always mean the types given by λ. (C, λ) is *compatible* with a tree t, if t is compatible with λ. If (C, λ) is compatible with another k-abstract context (C', λ'), then $C \cdot C'$ is also a k-abstract context. A *k-abstract loop* is a k-abstract-context where the $(k+1)$-type of the root induces the k-type of its hole. Notice that for a k-abstract loop C the set of $(k+1)$-types realised by nodes of C, and that realised by nodes of C^n is the same, for any $n \in \mathbb{N}_{\geq 1}$.

Let L be a tree language. Two trees s, t *agree on* L if either $s, t \in L$ or $s, t \notin L$. Two contexts C_1, C_2 are *congruent modulo* L, written $C_1 \cong_L C_2$, if for all contexts C and trees t, the trees $C \cdot C_1 \cdot t$ and $C \cdot C_2 \cdot t$ agree on L. A context C is *idempotent* if $C^2 \cong_L C$. A tree language is *regular*, if it is recognised by a *(bottom-up) tree automaton* (for a reference on tree automata, see e.g. [4]). The set of all contexts with the operation of concatenation forms a monoid. The quotient of this monoid by $\cong_L$ is called, in analogy to the word case, the *syntactic monoid* of L. Just as in the word case, a tree language is regular iff its syntactic monoid is finite. Therefore, with each regular tree language L come two associated constants: ω_L is the least number such that for each context C, C^{ω_L} is idempotent; κ_L is the least number such that, for each context C there exists a context C' of size at most κ_L with $C' \cong_L C$. In both cases, we usually omit the index L.

3 First-order logic with cardinality predicates

In this section, we consider an extension of first-order logic by *cardinality predicates*, and we characterise regular tree languages definable by this logic. Let $\mathrm{FO}_{\mathrm{card}}$ (resp. $\mathrm{FO}^m_{\mathrm{card}}$) denote the set of formulae of first-order logic that, in addition to the common rules for the formation of formulae of first-order logic, may use relation symbols from the set $\{\mathrm{C}_{a,m} : m \in \mathbb{N}_{\geq 1}, a \in [m)\}$, (resp. $\{\mathrm{C}_{a,m} : a \in [m)\}$) where each $C_{a,m}$ is a nullary relation symbol. The formula $\mathrm{C}_{a,m}$ shall be satisfied in a structure iff the size of the structure's universe is congruent a modulo m. A tree language L is $\mathrm{FO}_{\mathrm{card}}$-definable iff there exists an $\mathrm{FO}_{\mathrm{card}}$-sentence φ such that $t \in L$ iff $t \models \varphi$, for all trees t. For trees s, t, we write $s \approx^m_q t$ to denote that s and t agree on all tree languages definable by $\mathrm{FO}^m_{\mathrm{card}}$-sentences of quantifier depth at most q.

Our aim for this section will be a characterisation of the $\mathrm{FO}_{\mathrm{card}}$-definable tree languages in terms of their closure properties. To achieve this goal, we combine and extend the techniques developed in [2] and [9]. In [2], necessary and sufficient conditions for the FO-definability of a regular tree language were shown. To state the characterisation, we need to introduce the following notions. A tree language L is *aperiodic* iff there exists a constant $\ell \in \mathbb{N}$, such that $C^\ell \cong_L C^{\ell+1}$, for all contexts C.

▶ **Definition 3.1** (Guarded Swaps, [2]). Let t be a tree with root w.

Let $u \trianglelefteq v \trianglelefteq u' \trianglelefteq v'$ be nodes of t. Let $C := t[w, u), C_1 := t[u, v), D := t[v, u'), C_2 := t[u', v')$ be contexts and let $s := t_{|v'}$. The *vertical swap* of the tree $t = C \cdot C_1 \cdot D \cdot C_2 \cdot s$ between C_1 and C_2 is the tree $C \cdot C_2 \cdot D \cdot C_1 \cdot s$. If u and u' as well as v and v' are k-similar, for some $k \in \mathbb{N}$, then we say that the vertical swap is *k-guarded*.

Let u and v be incomparable nodes of t (i.e., neither $u \trianglelefteq v$ nor $v \trianglelefteq u$ holds). Let $C := t[w, u, v)$, and let $s_1 := t_{|u}$ and $s_2 := t_{|v}$. The *horizontal swap* of $t = C[s_1, s_2]$ between u and v is the tree $C[s_2, s_1]$. If u and v are k-similar, for some $k \in \mathbb{N}$, then we say that the horizontal swap is k-guarded.

A tree t' is a *k-guarded swap* of t iff it is either a k-guarded vertical swap or a k-guarded horizontal swap of t. A tree language L is *closed under k-guarded swaps* iff each tree t agrees on L with all its k-guarded swaps. L is closed under guarded swaps, if there exists a k such that L is closed under k-guarded swaps.

The characterisation of FO-definable tree languages by Benedikt and Segoufin reads as follows:

▶ **Theorem 3.2** ([2]). *A tree language is* FO*-definable iff it is regular, aperiodic, and closed under guarded swaps.*

For the special case of regular *word* languages it was shown in [9] that $\mathrm{FO}_{\mathrm{card}}$-definability of a regular language is characterised by certain closure properties as well. Let L be a regular word language over an alphabet Σ. The language L is said to be *closed under idempotent-guarded swaps* if for all words $\underline{p}, \underline{q}, r, e, f \in \Sigma^*$, such that e, f are idempotent it holds that $e\,\underline{p}\,f\,r\,e\,\underline{q}\,f \cong_L e\,\underline{q}\,f\,r\,e\,\underline{p}\,f$. A regular word language L is *closed under transfer* iff $x^{\omega+1}yz^\omega \cong_L x^\omega yz^{\omega+1}$, for all words x, y, z with $|x| = |z|$. The following was proved in [9]:

▶ **Theorem 3.3** ([9]). *A word language L is* $\mathrm{FO}_{\mathrm{card}}$*-definable iff it is regular, closed under idempotent-guarded swaps, and closed under transfer.*

The present section's goal is to show that Theorem 3.3 can be generalised to regular tree languages. To this end, we introduce a generalisation of the notion of closure under transfer to tree languages. Similarly to guarded swaps, it consists of a "vertical" and a "horizontal"

property. In this case, the vertical property is a direct translation of the notion of transfer from the syntactic monoid of word languages to the syntactic monoid of tree languages.

▶ **Definition 3.4** (Transfer). A regular tree language L is closed under *vertical transfer* if $C_1^{\omega+1} \cdot D \cdot C_2^{\omega} \cong_L C_1^{\omega} \cdot D \cdot C_2^{\omega+1}$ holds for all contexts C_1, D, C_2 with $|C_1| = |C_2|$. L is closed under *horizontal transfer* if the trees $C[C_1^{\omega+1} \cdot s_1,\ C_2^{\omega} \cdot s_2]$ and $C[C_1^{\omega} \cdot s_1,\ C_2^{\omega+1} \cdot s_2]$ agree on L, for all 2-contexts C, contexts C_1, C_2 with $|C_1| = |C_2|$, and trees s_1 and s_2. If L is closed under vertical and under horizontal transfer, then L is called *closed under transfer*.

The remainder of this section is devoted to the proof of the following theorem:

▶ **Theorem 3.5** (Characterisation of the $\mathrm{FO}_{\mathsf{card}}$-definable tree languages). *A tree language L is $\mathrm{FO}_{\mathrm{card}}$-definable iff it is regular, closed under guarded swaps, and closed under transfer.*

The transfer property, as stated in Definition 3.4, makes the connection with the corresponding property of word languages clear and will be useful when considering decidability questions in section 4. For the proof of Theorem 3.5, however, another formulation of transfer in terms of the following notion will be convenient:

▶ **Definition 3.6** (Growing a tree by a context; n-Template). Let t be a tree with root w, and let Δ be a context. Let p be a node of t, and let $C := t[w, p)$ and $s := t_{|p}$ (i.e. $t = Cs$). We say that the tree $C\Delta s$ is obtained from t by *letting t grow by Δ at p*.

For any $n \in \mathbb{N}$, an *n-template* is a tree T with n *expansion points*, i.e. n distinct distinguished nodes $p_1, \ldots, p_n$. We define $T\langle\rangle := T$ and, given a sequence of contexts $\Delta_1, \ldots, \Delta_\ell$, for an $\ell \leq n$, we let $T\langle\Delta_1, \ldots, \Delta_\ell\rangle$ be the tree obtained by letting $T\langle\Delta_1, \ldots, \Delta_{\ell-1}\rangle$ grow by Δ_ℓ at p_ℓ.

The following lemma gives an alternative formulation of transfer in terms of templates and is easily seen to be true:

▶ **Lemma 3.7** (Alternative formulation of transfer). *Let L be a regular tree language. L is* closed under transfer *iff for all 2-templates T and all contexts C_1, C_2 with $|C_1| = |C_2|$, the trees $T\langle C_1^{\omega+1}, C_2^{\omega}\rangle$ and $T\langle C_1^{\omega}, C_2^{\omega+1}\rangle$ agree on L.*

The outline of our proof of Theorem 3.5 is similar to the proof of Theorem 3.2 given in [2], in that a major part of it consists in the proof of the following lemma:

▶ **Lemma 3.8** (Main lemma). *Let L be a regular tree language that is closed under guarded swaps and closed under transfer. There exist $m, q \in \mathbb{N}$, such that L is a union of $\approx_q^m$-equivalence classes.*

Before we turn to the proof of this lemma, we show how to prove Theorem 3.5 with its help.

Proof of Theorem 3.5 using Lemma 3.8: For the "if-direction", let L be a regular tree language closed under transfer and guarded swaps. We want to show that L is $\mathrm{FO}_{\mathrm{card}}$-definable. By Lemma 3.8, we know that there exist $m, q \in \mathbb{N}$ such that L is a union of $\approx_q^m$-equivalence classes. It easy to see that each such class is definable by an $\mathrm{FO}_{\mathrm{card}}$-sentence, and the number of these classes is finite. Hence L can be defined by the disjunction of such sentences.

For the "only-if" direction, let L be an $\mathrm{FO}_{\mathrm{card}}$-definable tree language. For all $m \in \mathbb{N}_{\geq 1}$ and all $a \in [m)$, let $T_{a,m}$ denote the language of all trees of size a modulo m. We make use of the following easy observation:

▶ Claim 3.9. There exists an $m \in \mathbb{N}_{\geq 1}$ and FO-definable tree languages $L_0, \ldots, L_{m-1}$, such that $L = \bigcup_{a \in [m)} (L_a \cap T_{a,m})$.

Every FO-definable tree language is regular, as is each of the languages $T_{a,m}$. Hence Claim 3.9 immediately implies that L is regular, too. By Theorem 3.2, for each $a \in [m)$ there is a $k_a \in \mathbb{N}$ such that the language L_a is closed under k_a-guarded swaps. Let $k := \max\{k_0, \dots, k_{m-1}\}$. Each language L_a is obviously closed under k-guarded swaps, too. As guarded-swaps do not change the size of a tree, every language $L_a \cap T_{a,m}$ is closed under k-guarded-swaps, so their union L is so as well.

It remains to show closure of L under transfer. By Lemma 3.7, it suffices to show that for arbitrary 2-templates T, and contexts C_1 and C_2 with $|C_1| = |C_2|$, the trees $s := T\langle C_1^{\omega+1}, C_2^{\omega}\rangle$ and $t := T\langle C_1^{\omega}, C_2^{\omega+1}\rangle$ agree on L. For each $a \in [m)$, let φ_a be the FO-sentence defining L_a, and let q_a denote its quantifier depth. Let φ denote the $\mathrm{FO}_{\mathrm{card}}$-sentence defining L. If s, t agree on their size modulo m and on all sentences $\varphi_0, \dots, \varphi_{m-1}$, they must, by Claim 3.9, agree on φ as well. Let $q := \max\{q_0, \dots, q_{m-1}\}$. Because of the idempotency of C_1^{ω} and C_2^{ω}, the trees s and t agree on L iff for some $n \in \mathbb{N}_{\geq 1}$ the trees $s' := T\langle C_1^{n\omega+1}, C_2^{n\omega}\rangle$ and $t' := T\langle C_1^{n\omega}, C_2^{n\omega+1}\rangle$ agree on L. Note that, for any $\ell \in \mathbb{N}$, the number of occurrences of each ℓ-neighbourhood-type in the trees s' and t' is either the same (this is the case for the ℓ-neighbourhood-types of nodes whose ℓ-neighbourhood is neither strictly contained in the sequence of repetitions of C_1 nor in that of C_2) or can be made arbitrarily large by the choice of n (for ℓ-neighbourhood-types of nodes whose ℓ-neighbourhood is strictly contained in a long sequence of repetitions of C_1 or C_2). Thus we may deduce by an application of Hanf's Theorem (see e.g. [5]) that for some n, the trees s' and t' agree on all FO-sentences of quantifier depth at most q. By what was said above, this implies that the trees agree on φ. Therefore they agree on L, so L is closed under transfer. ◀

The remainder of section 3 is dedicated to the proof of the main lemma (Lemma 3.8).

Let L be a regular tree language that is closed under transfer and under k-guarded swaps, for a $k \in \mathbb{N}$. We want to show that there exist $q \in \mathbb{N}$ and $m \in \mathbb{N}_{\geq 1}$ such that two trees s, t that agree on all $\mathrm{FO}^m_{\mathrm{card}}$-sentences of quantifier depth q agree on L. We let m be given by the following lemma, which is an adaptation of a lemma proved in [9] for regular word languages. Its proof is a simple restatement of the proof given in [9] in terms of templates and contexts.

▶ **Lemma 3.10.** *Let L be a regular tree language. L is closed under transfer iff there exists an $m \in \mathbb{N}_{\geq 1}$, such that for all $\ell \in \mathbb{N}_{\geq 1}$, all contexts $\Delta_1, \dots, \Delta_\ell$, all ℓ-templates T and all $\delta_1, \dots, \delta_\ell \in \mathbb{N}$, if $\delta_1|\Delta_1| + \delta_2|\Delta_2| + \dots + \delta_\ell|\Delta_\ell| \equiv 0 \,(\mathrm{mod}\, m)$, then the trees $T\langle \Delta_1^{\omega}, \dots, \Delta_\ell^{\omega}\rangle$ and $T\langle \Delta_1^{\omega+\delta_1}, \dots, \Delta_\ell^{\omega+\delta_\ell}\rangle$ agree on L.*

As an intermediate step towards our goal, we show that $s \approx_q^m t$ implies that either t has the same number of occurrences of each $(k+1)$-type as s, and s and t agree on L (in this case we are done with the proof of the main lemma), or the number of occurrences of some type differs between s and t and, in this case, t agrees on L with "s with some additional contexts added". This is basically done as in the proof of Theorem 2 of [2], with only minor modifications of the lemmas used therein. Note, however, that in contrast to [2], we also have to care about the size of the trees modulo m. The following lemma gives a precise formulation of what we show:

▶ **Lemma 3.11.** *Let L be a regular tree language that is closed under transfer and k-guarded swaps, for a $k \in \mathbb{N}$. Let s, t be trees.*
(a) If $s =_{k+1} t$ and s, t are $(k+1)$-similar, then both trees agree on L.
(b) For all $d, m \in \mathbb{N}$ there exists a $q \in \mathbb{N}$ such that, if $s \approx_q^m t$ and not $s =_{k+1} t$, then there exists an $n \in \mathbb{N}_{\geq 1}$ and a sequence of k-abstract loops $(S_i)_{i\in(n]}$ and expansion points $p_1, \dots, p_n$ in s such that (i) $s\langle S_1, \dots, S_n\rangle$ agrees with t on L, (ii) $|s\langle S_1, \dots, S_n\rangle| \equiv |s| \,(\mathrm{mod}\, m)$, (iii) $|s|_\tau > d$, for all $(k+1)$-types τ occurring in $(S_i)_{i\in(n]}$.

We use Lemma 3.11(b) with $d := m\omega b$, where b is fixed according to Lemma 3.12 below. Now choose q according to Lemma 3.11(b), and let s, t be trees such that $s \approx_q^m t$. Our aim is to prove that s and t agree on L. If $s =_{k+1} t$, we are done due to Lemma 3.11(a). For the remainder of this section, assume that $s =_{k+1} t$ does not hold. Let $(S_i)_{i\in(n]}$ be given by Lemma 3.11(b). Let $t' := s\langle S_1, \ldots, S_n\rangle$. We know that t' and t agree on L, both trees have the same size modulo m, and each $(k+1)$-type occurring in $(S_i)_{i\in(n]}$ occurs strictly more than d times in s. We will construct a new tree s', $(k+1)$-similar to s, agreeing with s on L, and with all the $(k+1)$-types from the loops that distinguish s from t' added. I.e., we want to achieve $|s'|_\tau = |s|_\tau + |(S_i)_{i\in(n]}|_\tau = |t'|_\tau$ for all $(k+1)$-types τ. Then we are assured by Lemma 3.11(a) that t' and s' agree on L. Therefore, because t' and t as well as s' and s agree on L, we know that s and t agree on L, which is the conclusion that we are aiming at.

As a first step to construct s', we replace each loop S_i by a loop congruent to it modulo L, the size of which is bounded by a constant b depending only on L. The existence of such a loop is guaranteed by the following lemma, whose proof uses a standard pumping argument, where we have to ensure that the size of the given tree remains unchanged modulo m.

▶ **Lemma 3.12** (Loop bound). *Let $k \in \mathbb{N}$, $m \in \mathbb{N}_{\geq 1}$, and let L be a regular tree language. There exists a (computable) bound $b \in \mathbb{N}$ such that for all k-abstract loops Δ there exists a loop Δ' satisfying: (1) $\Delta' \cong_L \Delta$, (2) $|\Delta'| \leq b$, (3) $|\Delta'| \equiv |\Delta| \pmod m$, (4) $\Delta' \leq_{k+1} \Delta$.*

For each $i \in (n]$, let S_i' be the loop of size at most b given by the lemma for the loop S_i. Let $I := \{i_1, \ldots, i_\ell\} \subseteq [n]$ be a non-empty set of size at most m such that $|S_{i_1}'| + \cdots + |S_{i_\ell}'| \equiv 0 \pmod m$. Such a set exists by a simple application of the pigeonhole principle, because, as a consequence of Lemma 3.11(b), the summed size of the loops $(S_i)_{i\in(n]}$ is 0 modulo m. The next lemma tells us that there exists a tree, obtained from s by a sequence of k-guarded swaps, which contains (disjoint) copies of the loops $(S_i'^\omega)_{i\in I}$. The proof of the lemma uses Lemma 4 of [2] to include one loop after another into a tree obtained from s by k-guarded swaps (note that these change neither the $(k+1)$-type of the root nor the number of occurrences of $(k+1)$-types in a tree [2]), and then removes intersections between the images of the individual loops under the inclusion mappings by k-guarded swaps.

▶ **Lemma 3.13.** *Let L be a regular tree language closed under k-guarded swaps, for $k \in \mathbb{N}$. Let $(\Delta_i)_{i\in(\ell]}$, for $\ell \in \mathbb{N}_{\geq 1}$, be a sequence of k-abstract loops. For all trees s such that $(\Delta_i)_{i\in(\ell]} <_{k+1} s$, there exists an ℓ-template T such that $T\langle\Delta_1, \ldots, \Delta_\ell\rangle$ agrees with s on L, $T\langle\Delta_1, \ldots, \Delta_\ell\rangle =_{k+1} s$, and $T\langle\Delta_1, \ldots, \Delta_\ell\rangle$ is $(k+1)$-similar to s.*

Recall that we know, by Lemma 3.11 and 3.12, that each $(k+1)$-type occurring in one of the loops $(S_i')_{i\in(n]}$ occurs more than d times in s. The contexts $(S_i')_{i\in(n]}$ being k-abstract loops, we do not introduce any *new* $(k+1)$-types when taking their ω-powers. Hence, each $(k+1)$-type of $(S_i'^\omega)_{i\in I}$ occurs at least d times in s.

We want to apply Lemma 3.13 for $(\Delta_i)_{i\in(\ell]} := (S_i'^\omega)_{i\in I}$ and s. To do this, we need to make sure that $(S_i')_{i\in I} <_{k+1} s$. This is assured by taking $d := m\omega b$ as, obviously, there cannot be more occurrences of any particular $(k+1)$-type in $(S_i')_{i\in I}$ than there are nodes in $(S_i')_{i\in I}$ altogether. Let T be given by Lemma 3.13. By Lemma 3.10, we know that $T\langle S_{i_1}'^\omega, \ldots, S_{i_\ell}'^\omega\rangle$ agrees with $T\langle S_{i_1}'^\omega \cdot S_{i_1}', \ldots, S_{i_\ell}'^\omega \cdot S_{i_\ell}'\rangle$ on L. This tree, in turn, agrees with $T\langle S_{i_1}'^\omega \cdot S_{i_1}, \ldots, S_{i_\ell}'^\omega \cdot S_{i_\ell}\rangle$ on L, as each context S_i' is congruent S_i modulo L by Lemma 3.12. By this reasoning, we have added a copy of $(S_i)_{i\in I}$ (and hence, especially, a copy of every $(k+1)$-type therein) to a tree that agrees with s on L.

Now we may apply the same argument successively on the tree just obtained to add the remaining $(k+1)$-types from $\{S_1, \ldots, S_n\} \setminus \{S_{i_1}, \ldots, S_{i_\ell}\}$. Finally this yields the desired tree s'. This finishes the proof of Lemma 3.8. ◀

4 Decidability

In this section we show that it is possible to decide if a given regular tree language is closed under transfer. Combined with the decidability of closure under guarded swaps (see [2]) and our results from section 3 this implies that $\mathrm{FO}_{\mathrm{card}}$-definability of a given regular tree language is decidable.

▶ **Theorem 4.1.** *It is decidable whether the language L recognised by a given tree automaton $\mathcal{A}$ is closed under transfer.*

Proof. We assume w.l.o.g. that $\mathcal{A}$ is a minimal deterministic tree automaton with state set Q. In order to decide whether L is closed under horizontal transfer, we will check all possible counter examples to this property. If L does *not* posses the closure property, there exists a 2-context C and contexts Δ_1, Δ_2 with $|\Delta_1| = |\Delta_2|$ and trees s_1, s_2 such that the trees $t_1 := C[\Delta_1^{\omega+1} s_1, \Delta_2^{\omega} s_2]$ and $t_2 := C[\Delta_1^{\omega} s_1, \Delta_2^{\omega+1} s_2]$ do *not* agree on L. Let $f_{\Delta_1}, f_{\Delta_2} : Q \to Q$ and $f_C : Q \times Q \to Q$ denote the transition functions induced by $\mathcal{A}$ and, respectively, Δ_1, Δ_2, and C on Q. The trees t_1 and t_2 do not agree on L iff there exist states p_1, p_2 and q_1^+,q_1,q_2, q_2^+ such that: (1) $f^{\omega}_{\Delta_1}(p_1) = q_1$ and $f^{\omega+1}_{\Delta_1}(p_1) = q_1^+$, (2) $f^{\omega}_{\Delta_2}(p_2) = q_2$ and $f^{\omega+1}_{\Delta_2}(p_2) = q_2^+$, (3) $f_C(q_1^+, q_2) \neq f_C(q_1, q_2^+)$.

Let $R \subseteq Q^6$ be a relation such that a tuple of states $\vec{q} := (p_1, p_2, q_1^+, q_1, q_2, q_2^+)$ belongs to R, iff there are contexts Δ_1, Δ_2 with $|\Delta_1| = |\Delta_2|$ satisfying conditions (1) and (2) above.

For all $i \in \mathbb{N}$, let M_i denote the set of transition functions induced by contexts of size i on Q. The set M_i can be computed by simply enumerating all of the (finitely many) contexts of size i and computing the transition function of each such context in turn. Hence, we can recursively enumerate R by iterating through all the sets M_i and comparing the behaviour of the transition functions therein upon all combinations of states. By a pumping argument one sees that there exists a computable bound n such that, if there are contexts Δ_1, Δ_2 witnessing conditions (1) and (2) for a tuple $\vec{q}$, then there have to be such witnesses of size at most n. Hence, R is decidable.

Now we can decide closure under horizontal transfer by checking all possible counter examples: For all 6-tuples $\vec{q}$ as above with $\vec{q} \in R$, we compute all possible transition functions $f : Q \times Q \to Q$ induced by $\mathcal{A}$ and check if $f(q_1^+, q_2) \neq f(q_1, q_2^+)$. If such an f is found, condition (3) is satisfied and L cannot be closed under horizontal transfer. On the other hand, if the check fails for all functions f, we know that L is closed under horizontal transfer.

The decidability of closure under vertical transfer follows using an analogous argument. ◀

Combining Theorem 3.5 with Theorem 4.1 and the decidability of closure under guarded swaps obtained in [2], immediately leads to:

▶ **Corollary 4.2.** *It is decidable whether the language L recognised by a given tree automaton $\mathcal{A}$ is $\mathrm{FO}_{\mathrm{card}}$-definable.*

5 Addition-invariant FO

The set of all first-order formulae that may use the additional binary relation symbol $<$ and a ternary relation symbol $+$ is denoted by $\mathrm{FO}[<,+]$. A $\{<,+\}$-expansion of a tree t is a structure that keeps the interpretation of P_a (for all $a \in \Sigma$) and $\mathrm{S}_1, \mathrm{S}_2$ given by t, and interprets $<$ as a linear order on t and $+$ as the addition relation induced by $<$. A $\mathrm{FO}[<,+]$-formula φ is *addition-invariant*, if for all $\{<,+\}$-expansions of s, s' of a tree, $s \models \varphi$ iff $s' \models \varphi$. Let $+$–inv–FO denote the set of addition-invariant formulae. This section's main result is the following theorem, generalising a result of [9] from words to trees:

▶ **Theorem 5.1.** *Let L be a regular tree language. The following statements are equivalent: (1) L is* +–inv–FO*-definable, (2) L is closed under transfer and guarded swaps, (3) L is* $\mathrm{FO}_{\mathrm{card}}$*-definable.*

The equivalence of statements (3) and (2) was proved in Theorem 3.5. It is easily seen that any regular tree language definable by an $\mathrm{FO}_{\mathrm{card}}$-sentence φ is definable by an +–inv–FO-sentence. For example, the following +–inv–FO-sentence defines $\mathrm{C}_{1,2}$ (where we assume that the least element with respect to $<$ has index 0):

$$\exists x\, \exists z\, \big(z = x + x \;\wedge\; \neg \exists y\, (y < z)\big).$$

For the remainder of this section, we will be occupied by the proof that +–inv–FO-definability implies closure under guarded swaps and transfer. The following proofs make extensive use of *first-order interpretations*; see e.g. [5] for an exposition of this technique.

Closure under guarded swaps. To prove the closure of +–inv–FO-definable regular tree languages under guarded swaps, we use the following Lemma 5.2, which is an immediate consequence of [8, Proposition 6.11] (which lies at the heart of the results from [9], too). To state the lemma, we need the following notations: Let σ be a relational signature. For each σ-structure A, we write σ^A for the set of relations of A. Let A, B be σ-structures. Let α be a mapping from the universe of A to the universe of B. For a relation $R \in \sigma^A$ of arity m, we define $\alpha(R) := \{(\alpha(a_1), \ldots, \alpha(a_m)) : (a_1, \ldots, a_m) \in R\}$. For $\sigma^A = \{R_1, \ldots, R_n\}$, let $\alpha(\sigma^A) := \{\alpha(R_1), \ldots, \alpha(R_n)\}$. We write $A \approx_q B$ to indicate that A and B satisfy the same first-order-sentences of quantifier depth at most q.

▶ **Lemma 5.2** ([8])**.** *Let $q'', h, e \in \mathbb{N}_{\geq 1}$ and let σ be a signature. There exists an infinite set $P := \{p_1 < p_2 < p_3 \ldots\} \subseteq \mathbb{N}$ with $p_1 > h$ and $p_i \equiv h \,(\mathrm{mod}\, e)$, for all $i \in \mathbb{N}_{\geq 1}$, and a number q' such that the following is true for all finite σ-structures M and all linear orders $<_1$ and $<_2$ on M's universe: if $\langle M, <_1\rangle \approx_{q'} \langle M, <_2\rangle$, then $\langle \mathbb{Z}, +, P, \alpha_1(\sigma^M)\rangle \approx_{q''} \langle \mathbb{Z}, +, P, \alpha_2(\sigma^M)\rangle$, where α_i is a map taking the j-th node of M according to $<_i$ to p_j, for $i \in (2]$ and $j \in \mathbb{N}_{\geq 1}$.*

The second ingredient to our proof of the closure of +–inv–FO-definable tree languages under guarded swaps is a lemma of [6] which was used in [3] to prove closure under guarded swaps of order-invariantly definable tree languages:

▶ **Lemma 5.3** (implicit in [6])**.** *Let $x, q' \in \mathbb{N}$, and let σ be a signature. There exists $k' \in \mathbb{N}$ such that for each finite σ-structure M and all x-tuples $\bar{a}$ and $\bar{b}$ of M with isomorphic k'-neighbourhoods, there exist linear orders $<_1$ and $<_2$ of the universe of M, whose initial elements are respectively $\bar{a}\bar{b}$ and $\bar{b}\bar{a}$, such that $\langle M, <_1\rangle \approx_{q'} \langle M, <_2\rangle$.*

We use the lemmas 5.2 and 5.3 together with an interpretation argument, to prove:

▶ **Lemma 5.4.** *Let L be a regular tree language. If L is definable by an* +–inv–FO*-sentence, then L is closed under guarded horizontal swaps.*

Proof sketch. Let φ be an +–inv–FO-sentence defining L. Let Q be the state set of a minimal deterministic tree automaton recognising L. We want to show that L is closed under k-guarded horizontal swaps, for a $k \in \mathbb{N}$ that will be fixed later on. Consider a tree t with incomparable k-similar nodes u and v. Let $t_1 := t_{|u}$ and $t_2 := t_{|v}$, i.e. $t = C[t_1, t_2]$ for a 2-context C. Let $t' := C[t_2, t_1]$. We may assume that the trees t_1 and t_2 have height at least k. Taking $k > \kappa_L k' + |Q^Q|$ (with κ_L as defined at the end of section 2), where k' will be fixed later on by our application of Lemma 5.3, a standard pumping argument shows that we may

assume that $t_1 = DEt'_1$, for a tree t'_1 and contexts E, D such that E is idempotent, and D is $\kappa_L k'$-similar to t_2. Let $e := |E|$. Without loss of generality, $e \leq \kappa_L$ (if not, we can replace E by a congruent context of that size) and $|t'_1| \geq e$ (if not, we can prepend a copy of E to it).

For $i \in \{1, 2\}$, we decompose t_i into *blocks* of size e, plus a *residual block* of size $n_i := |t_i| \bmod e$, if $|t_i|$ is not divisible by e: A block consists of e consecutive nodes of t_i, ordered according to the *bf-order* of the tree (cf., section 2). We let M be a structure using the set of blocks of size e of t_1 and t_2 as universe, with relations which encode the following information about t_1 and t_2: the successor relations between the nodes of the different blocks (resp., between the blocks and the residual blocks not in M), the position of E, and the labels of the nodes in each block. Let b_1 and b_2 be the blocks containing the roots of t_1 and t_2, respectively. Since t_1 and t_2 are k-similar, the k/e-neighbourhoods of b_1 and b_2 in M are isomorphic. We let k' be given by Lemma 5.3 for $x := 1$ and a q' to be fixed later on. By our choice of k we have $k' \leq k/e \leq k/\kappa_L$. By Lemma 5.3 we obtain two linear orders $<_1$ and $<_2$ on M such that, according to $<_1$, b_1 comes first and b_2 comes second, and according to $<_2$ it is just the other way round. Lemma 5.3 guarantees that $\langle M, <_1\rangle$ and $\langle M, <_2\rangle$ agree on all FO[$<, +$]-sentences of quantifier depth $\leq q'$. Thus, by Lemma 5.2, we obtain structures M_1 and M_2 over the integers which contain "stretched copies" of $\langle M, <_1\rangle$ and $\langle M, <_2\rangle$, respectively. I.e. some elements of M_1 and M_2, marked by a unary predicate P, correspond to the original structures, and other positions in between do not. The number h of Lemma 5.2 is set to be $|C| + n_1 + n_2$, and q'' will be fixed later on. We choose q' as given by Lemma 5.2 and obtain that M_1 and M_2 agree on all FO[$<, +$]-sentences of quantifier depth at most q''.

Now we specify an FO[$<, +$]-interpretation that transforms M_1 into a tree agreeing with t on L, and M_2 into a tree agreeing with t' on L: The set of nodes of the trees consist of all non-negative integers before the least position in P that is not included in any "stretched relation". The successor relations and labels of the first e nodes starting at a node p where P holds are interpreted in such a way that t_1 and t_2 are simulated on these nodes; for this, the "stretched copies" of the relations from $\langle M, <_1\rangle$ resp. $\langle M, <_2\rangle$ are used. The nodes between $p + e$ and the next number p' in P, are interpreted as copies of the idempotent context E; the same is done for the nodes between the positions h and $p_1 - 1$. All these copies of E are inserted at the original position of E in the simulated tree t_1. In the first h nodes, the (inner tree of) the 2-context C and the two residual blocks of size n_1 and n_2 are simulated. The simulated parent of the first hole of C is linked to the node that is simulated at the first position in P; the parent of the second hole of C is linked to the node at the second position in P. This way, for $\widetilde{t_1} := DE^i t'_1$, for a suitable $i \in \mathbb{N}_{\geq 1}$, the interpretation turns M_1 into the tree $C[\widetilde{t_1}, t_2]$ (which, as E is idempotent, agrees with t on L), and M_2 into $C[t_2, \widetilde{t_1}]$ (which agrees with t' on L). By choosing q'' larger than the sum of the maximal quantifier depth of the formulae of this interpretation, and the quantifier depth of the formula φ defining the language L, we ensure that t and t' agree on φ, finishing the proof. ◀

Our next goal is to prove that every +–inv–FO-definable regular tree language is closed under guarded vertical swaps as well. To achieve this, we first prove the closure under a variant of guarded vertical swaps, where the guardedness assumptions are somewhat strengthened: A language L is said to be *closed under strongly-k-guarded* vertical swaps, for $k \in \mathbb{N}$, if each tree t containing nodes $u \lhd v \lhd u' \lhd v'$, such that u and u' are k-similar, v and v' have *isomorphic k-neighbourhoods*, and the k-neighbourhoods of v and v' and k-spills of u and u' are all mutually disjoint, agrees on L with its vertical swap between $t[u, v)$ and $t[u', v')$.

▶ **Lemma 5.5.** *Let L be a regular tree language. If L is $+$–inv–FO-definable, then L is closed under strongly-k-guarded vertical swaps, for some $k \in \mathbb{N}$.*

The proof of Lemma 5.5 proceeds similarly to the proof of Lemma 5.4 (the *strongly*-guarded swaps being necessary to apply Lemma 5.3). We continue by showing that being closed under strongly-guarded vertical swaps is actually equivalent to being closed under guarded vertical swaps, if the language under consideration is closed under guarded horizontal swaps, too.

▶ **Lemma 5.6.** *Let L be a tree language. L is closed under guarded swaps iff it is closed under strongly-guarded vertical swaps and guarded horizontal swaps.*

Proof idea. Closure under guarded swaps immediately implies closure under strongly-guarded swaps and guarded horizontal swaps. Let L be a tree language that is closed under strongly-k'-guarded vertical swaps and k'-guarded horizontal swaps. We show that L is closed under k-guarded vertical swaps, for a suitable number $k > k'$. Let $t := C\Delta_1\Delta\Delta_2 s$ be given as in the definition of vertical guarded-swaps, and let $t' := C\Delta_2\Delta\Delta_1 s$. The proof proceeds by distinguishing cases depending on the root-hole-distance of the contexts Δ_1, Δ, Δ_2. We show how to find nodes $\tilde{u}$ and $\tilde{u}'$ in the k-spills of the root of Δ_1 resp. Δ_2, and $\tilde{v}$ and $\tilde{v}'$ in the k-spills of the hole of Δ_1 resp. Δ_2 in t, fulfilling the preconditions for a strongly-k'-guarded vertical swap between $t[\tilde{u}, \tilde{v})$ and $t[\tilde{u}', \tilde{v}')$. After this swap, we have either swapped "too much" or "not enough" of Δ_1 and Δ_2. In these cases, we are able to repair the remaining parts by a series of k'-guarded horizontal swaps between the (incomparable) nodes "around" the nodes swapped in the strongly-guarded vertical swap. ◀

Closure under transfer. We now show that a regular tree language definable by an $+$–inv–FO-sentence is closed under transfer. This is the easier part of the proof of Theorem 5.1, as we are able to build directly on results proved in [9]. To state the according result, we need some further notation: Given words $w, x \in \Sigma^*$, with $|w| \geq |x|$, we denote by $|w|_x$ the number of non-overlapping occurrences of x as a factor in w. Furthermore we say that a sentence φ *separates* languages L_1 and L_2 iff every word in L_1, but no word in L_2, satisfies φ.

▶ **Lemma 5.7** (Proposition 3.3 of [9])**.** *Let $n \in \mathbb{N}$ with $n \geq 2$, $y \in \Sigma^*$, $\bar{x} \in (\Sigma \times \{1\})^*$ and $\bar{z} \in (\Sigma \times \{2\})^*$. For all $a, b \in \mathbb{N}$, let*

$$L_{n,a,b} := \{w \in y\bar{x}(\bar{x}\bar{z}|\bar{z}\bar{z})^* : |w|_{\bar{x}}, |w|_{\bar{z}} \geq n, |w|_{\bar{x}} \equiv a \,(\mathrm{mod}\, n), |w|_{\bar{z}} \equiv b \,(\mathrm{mod}\, n)\}.$$

There exists no $\mathrm{FO}[<,+]$-sentence that separates $L_{n,1,0}$ from $L_{n,0,1}$.

We use Lemma 5.7 and proceed with a proof by contradiction: If a regular tree language L is not closed under transfer, we show by an interpretation argument that there exists a $\mathrm{FO}[<,+]$-sentence separating a suitable word language $L_{n,1,0}$ from a word language $L_{n,0,1}$ for $n := \omega_L$. This is akin to what is done in [9] to prove that every regular *word* language is closed under transfer, the difference being that we have to simulate trees in words.

▶ **Lemma 5.8.** *Let L be a regular tree language. If L is definable by an $+$–inv–FO-sentence, then L is closed under transfer.*

Proof sketch. Assume that L is not closed under transfer. This means, there are contexts Δ_1, Δ_2 with $|\Delta_1| = |\Delta_2|$ and a 2-template T, such that $t := T\langle\Delta_1^{\omega+1}, \Delta_2^{\omega}\rangle \in L$, but $t' := T\langle\Delta_1^{\omega}, \Delta_2^{\omega+1}\rangle \notin L$. Let $t_{i,j} := T\langle\Delta_1^i, \Delta_2^j\rangle$, for all $i, j \in \mathbb{N}_{\geq 1}$. Because Δ_1^{ω} and Δ_2^{ω} are idempotent, we may repeat both contexts in the trees t and t' without affecting membership

in L. Hence, for all $i, j \geq \omega$, (1) if $i \equiv 1 \,(\mathrm{mod}\, \omega), j \equiv 0 \,(\mathrm{mod}\, \omega)$, then $t_{i,j} \in L$, and (2) if $i \equiv 0 \,(\mathrm{mod}\, \omega), j \equiv 1 \,(\mathrm{mod}\, \omega)$, then $t_{i,j} \notin L$. As we are aiming at a contradiction to Lemma 5.7, we fix the numbers and words therein: We take n to be ω. To each tree we assign the word of its labels, ordered according to the bf-order on the nodes of the tree. Let y be the word obtained from the template T in that way. Let x and z be the words obtained from the inner tree of the contexts Δ_1 and Δ_2, respectively. Let $\bar{x}$ be the word x with each symbol $a \in \Sigma$ that occurs in x replaced by the tuple $(a, 1)$, and let $\bar{z}$ be obtained from z by tagging each symbol of z accordingly by 2. Let $L_T := \{t_{i,j} : i, j \in \mathbb{N}\}$. We define an FO[$<, +$]-interpretation that interprets trees from L_T in words from the language $y\bar{x}(\bar{x}\bar{z}|\bar{z}\bar{z})^*$ such that, given a word $w \in y\bar{x}(\bar{x}\bar{z}|\bar{z}\bar{z})^*$ with $i := |w|_{\bar{x}}$ and $j := |w|_{\bar{z}}$, this interpretation constructs the tree $t_{i,j}$. It is possible to do this by FO[$<, +$]-formulae, because Δ_1, Δ_2, and T are fixed, and we can use the tags 1 and 2 in the words $\bar{x}$ and $\bar{z}$ to identify the positions in a word where subwords corresponding to Δ_1 resp. Δ_2 start. Now consider the languages $L_{n,1,0}$ and $L_{n,0,1}$ (for $n := \omega$) of Lemma 5.7: If $w \in L_{n,1,0}$, then, by (1), $t_{i,j} \in L$. On the other hand, if $w \in L_{n,0,1}$, then $t_{i,j} \notin L$. Let φ be the $+$–inv–FO-sentence defining L. We alter φ according to our interpretation to obtain an FO[$<, +$]-sentence φ'. By the addition-invariance of φ, the choice of the addition relation (here, the one induced by the linear order on the word) is immaterial for the satisfaction of φ by $t_{i,j}$. Therefore, $w \models \varphi'$ iff $t_{i,j} \models \varphi$ iff $t_{i,j} \in L$. Thus, φ' separates $L_{n,0,1}$ from $L_{n,1,0}$, contradicting Lemma 5.7 and finishing the proof of Lemma 5.8. ◀

From the lemmas 5.8, 5.4, 5.5, 5.6, we obtain that every $+$–inv–FO-definable regular tree language is closed under transfer and guarded swaps, concluding the proof of Theorem 5.1. ◀

Acknowledgement. We thank Luc Segoufin for helpful discussions on the subject of this paper.

References

1 D. Beauquier and J.-E. Pin. Factors of words. In *Proc. ICALP'89*, volume 372 of *Lecture Notes in Computer Science*, pages 63–79. Springer-Verlag, 1989.

2 M. Benedikt and L. Segoufin. Regular tree languages definable in FO and in FO$_{mod}$. *ACM Transactions on Computational Logic*, 11(1), 2009.

3 M. Benedikt and L. Segoufin. Towards a characterization of order-invariant queries over tame graphs. *Journal of Symbolic Logic*, 74(1):168–186, 2009.

4 H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available at `http://tata.gforge.inria.fr/`. Release: October, 12th 2007.

5 H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Springer-Verlag, 2nd edition, 1999.

6 M. Grohe and T. Schwentick. Locality of order-invariant first-order formulas. *ACM Transactions on Computational Logic*, 1(1):112–130, 2000.

7 L. Libkin. *Elements of Finite Model Theory*. Springer-Verlag, 2004.

8 N. Schweikardt. An Ehrenfeucht-Fraïssé game approach to collapse results in database theory. *Information and Computation*, 205(3):311–379, 2007.

9 N. Schweikardt and L. Segoufin. Addition-invariant FO and regularity. In *Proc. 25th IEEE Symposium on Logic in Computer Science (LICS'10)*, pages 285–294. IEEE, 2010.

10 M.P. Schützenberger. On finite monoids having only trivial subgroups. *Information and Control*, 8(2):190–194, 1965.

11 H. Straubing. *Finite automata, formal logic, and circuit complexity*. Birkäuser, 1994.

Simpler Approximation of the Maximum Asymmetric Traveling Salesman Problem

Katarzyna Paluch*[1], Khaled Elbassioni[2], and Anke van Zuylen[2]

1 **Institute of Computer Science, University of Wroclaw**
ul. Joliot-Curie 15, room 304, 50-383 Wroclaw, Poland
abraka@cs.uni.wroc.pl

2 **Max Planck Institute for Informatics**
Campus E 13, Saarbrücken, Germany
{elbassio,anke}@mpi-inf.mpg.de

Abstract

We give a very simple approximation algorithm for the maximum asymmetric traveling salesman problem. The approximation guarantee of our algorithm is 2/3, which matches the best known approximation guarantee by Kaplan, Lewenstein, Shafrir and Sviridenko. Our algorithm is simple to analyze, and contrary to previous approaches, which need an optimal solution to a linear program, our algorithm is combinatorial and only uses maximum weight perfect matching algorithm.

1998 ACM Subject Classification I.1.2 Algorithms

Keywords and phrases approximation algorithm, maximum asymmetric traveling salesman problem

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.501

1 Introduction

In this paper, we study the maximum asymmetric traveling salesman problem (MAX ATSP). The input to the MAX ATSP is a directed complete graph $G = (V, A)$ with edge weights $w(e) \geq 0$ for all $e = (i, j) \in A$. The objective is to find a tour of maximum weight. This problem is known to be APX-hard [11], and research has focused on finding good approximation algorithms for this problem. Good approximation algorithms are of particular interest because they imply good approximations for a number of related problems. For example, it was shown by Breslauer, Jiang and Jiang [3] that an α-approximation algorithm for MAX ATSP implies a $(\frac{7}{2} - \frac{3}{2}\alpha)$-approximation algorithm for the shortest superstring problem; a problem that arises in DNA sequencing and data compression. Any α-approximation algorithm for MAX ATSP implies an algorithm with the same guarantee for the maximal compression problem defined by Tarhio and Ukkonen [12].

The current best approximation algorithm for MAX ATSP is due to Kaplan, Lewenstein, Shafrir and Sviridenko [5] and achieves an approximation guarantee of $\frac{2}{3}$. Their algorithm needs the optimal solution of an LP relaxation of the max ATSP, which is scaled up to an integral solution by multiplying it by the least common denominator of all variables. From the scaled up solution, a pair of cycle covers is extracted that have a combined weight of at least twice the weight of the optimum solution. Finally the obtained pair of cycle covers

* Supported by MNiSW grant number N N206 368839, 2010-2013.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 501–506

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

is processed using a rather complicated coloring lemma. Previous results on MAX ATSP appeared among others in [4], [6] [1], [9]. In this paper, we give a very simple approximation algorithm that achieves the same guarantee as the algorithm by Kaplan et al. Our algorithm is combinatorial in nature: it constructs a certain matching instance and computes a maximum weight matching. We then give a simple procedure that uses this matching to form three tours, and we show that the average weight of these tours is at least $\frac{2}{3}$ times the optimum.

Other variants of the maximum traveling salesman problem that have been considered are among others: the maximum symmetric traveling salesman problem (MAX TSP), in which the underlying graph is undirected - currently the best known approximation ratio is $\frac{7}{9}$ [10], the maximum metric symmetric traveling salesman problem, in which the edge weights satisfy the triangle inequality - the best approximation factor is $\frac{7}{8}$ [7], the maximum asymmetric traveling salesman problem with triangle inequality - the best approximation ratio is $\frac{35}{44}$ [8].

The key idea in our approach to MAX ATSP is the following. A natural first idea that has been very fruitful when designing an approximation algorithm for maximum traveling salesman problems is to start with a *cycle cover* of maximum weight, i.e., a maximum weight collection of edges such that each node is incident to exactly two edges (in the undirected case) or exactly one incoming and one outgoing edge (in the directed case). Such a cycle cover can be found in polynomial time by a reduction to maximum weight matching. Since the optimal tour is a cycle cover, the weight of the maximum weight cycle cover is at least the weight of the optimal tour. By removing the lightest edge from each cycle, we obtain a collection of node disjoint paths, which can be arbitrarily connected to form a tour. For the asymmetric case, this approach will give a $\frac{1}{2}$-approximation algorithm, since the cycle cover may contain cycles of length two (2-cycles). If we could find a maximum weight cycle cover without 2-cycles, then we would achieve a $\frac{2}{3}$-approximation, but, unfortunately, finding a maximum weight cycle cover without 2-cycles is APX-hard [2].

Our key observation is the fact that we can exclude 2-cycles from the cycle cover, if we allow the cycle cover to contain "half edges": We split each edge (i, j) into two half edges; the head of (i, j) (which is incident to j but not to i) and the tail of (i, j) (which is incident to i, but not to j). A half edge has weight equal to half the weight of the original edge. Now, for a pair of edges $(i, j), (j, i)$, we ensure that the solution does not contain both edges, but we do allow the solution to contain the heads of both (i, j) and (j, i) or the tails of both (i, j) and (j, i). We show that such a cycle cover with half edges can be computed by an appropriate reduction to a maximum weight perfect matching problem. Finally, we show how to use the cycle cover with half edges to extract three tours with total weight at least twice the weight of the cycle cover.

2 Cycle Covers without 2-Cycles but with Half-Edges

We begin by introducing the maximum weight cycle cover problem without 2-cycles, but with half-edges, and showing it can be computed in polynomial time. Given a complete directed graph $G = (V, E)$ and weights $w(e) \geq 0$ for each $e \in E$, a cycle cover is a subset C of the edges, so that each $i \in V$ has exactly one outgoing and one incoming edge in C. In the maximum weight cycle cover problem with half-edges, we allow the solution C to contain "only the head or only the tail" of an edge (i, j). Such a half-edge has weight $\frac{1}{2}w(i, j)$ and is thought to be incident to only one endpoint of the edge (i, j). We introduce these half-edges so that we can ensure that our cycle cover does not have 2-cycles. This gives rise to the following problem:

▶ **Definition 1.** Given a directed graph $G = (V, E)$ with edge weights $w(i, j) \geq 0$ for every $(i, j) \in E$, let $\tilde{G} = (\tilde{V}, \tilde{E})$ be the graph obtained from G by replacing each $(i, j) \in E$ by a node $v_{(i,j)}$ and two edges $(i, v_{(i,j)})$ and $(v_{(i,j)}, j)$, each with weight $\frac{1}{2}w(i, j)$. A *cycle cover without 2-cycles but with half-edges* is a subset $\tilde{C} \subseteq \tilde{E}$ such that

(i) each node in V has exactly one outgoing and one incoming edge in $\tilde{C}$;
(ii) for each $(i, j) \in E$, $\tilde{C}$ contains either zero edges from $\{(i, v_{(i,j)}), (v_{(i,j)}, j), (j, v_{(j,i)}), (v_{(j,i)}, i)\}$, or it contains exactly one edge incident to i and one edge incident to j.

▶ **Lemma 2.** *We can find a maximum weight cycle cover without 2-cycles but with half-edges in polynomial time.*

Proof. We reduce the problem of finding $\tilde{C}$ of maximum weight to a maximum weight perfect matching problem in the following undirected graph G':

For each node $i \in V$, we create two nodes, in_i and out_i. For an edge $e = (i, j) \in E$, we create two nodes $\text{head}_{(i,j)}$ and $\text{tail}_{(i,j)}$, and we have three undirected edges $\{\text{out}_i, \text{tail}_{(i,j)}\}, \{\text{tail}_{(i,j)}, \text{head}_{(i,j)}\}$ and $\{\text{head}_{(i,j)}, \text{in}_j\}$. The weight of the first and third edge is $\frac{1}{2}w(i, j)$, and the weight of the second edge is 0.

Note that a perfect matching in this graph corresponds to a cycle cover of G of the same weight and vice versa: for each edge (i, j) of G, a perfect matching of G' either contains edge $\{\text{tail}_{(i,j)}, \text{head}_{(i,j)}\}$ or edges $\{\text{out}_i, \text{tail}_{(i,j)}\}, \{\text{head}_{(i,j)}, \text{in}_j\}$. Of course, if we are only interested in computing a cycle cover of G, then it suffices to split each node i into two: in_i and out_i and connect in_j and out_i via an edge whenever $(i, j) \in G$ and compute a perfect matching in the thus obtained bipartite undirected graph. We will now show how to modify G' so that a perfect matching in G' corresponds to a maximum weight cycle cover without 2-cycles but with half-edges.

A perfect matching in G' defines a set of half-edges $\tilde{C} \subseteq \tilde{E}$ of the same weight that satisfies property (i) in Definition 1, but $\tilde{C}$ may contain four half-edges that correspond to a 2-cycle in G. To enforce that $\tilde{C}$ also satisfies property (ii), we add two additional nodes, $i_{\{i,j\}}$ and $j_{\{i,j\}}$ for each pair of edges $(i, j), (j, i)$. We add an edge from $i_{\{i,j\}}$ to $\text{tail}_{(i,j)}$ and $\text{head}_{(j,i)}$, and we add an edge from $j_{\{i,j\}}$ to $\text{head}_{(i,j)}$ and $\text{tail}_{(j,i)}$. These edges all have weight 0. The fact that the additional nodes need to be matched ensures that $\tilde{C}$ does not contain all four half-edges corresponding to (i, j) and (j, i).

For a pair of edges $(i, j), (j, i)$ in G, the matching instance G' thus has a gadget containing 10 edges. See Figure 1. We now verify that a perfect matching in G' corresponds to a cycle

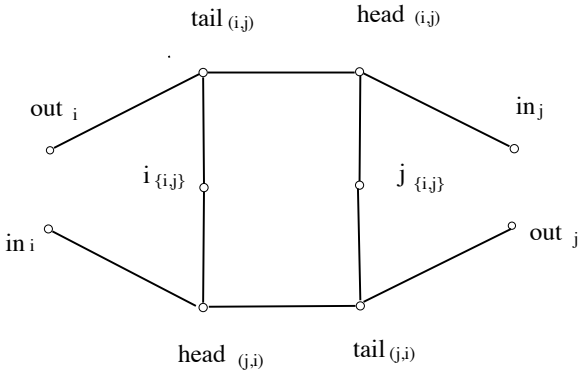


Figure 1 A gadget corresponding to a 2-cycle on vertices i and j.

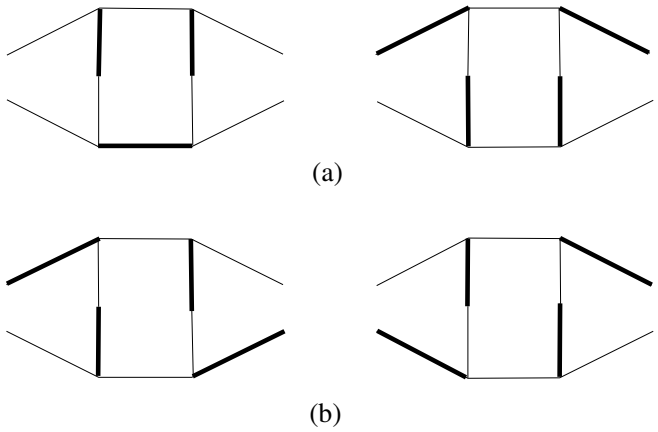


Figure 2 Possible ways of matching vertices $i_{\{i,j\}}, j_{\{i,j\}}, \text{head}_{(i,j)}, \text{tail}_{(i,j)}, \text{head}_{(j,i)}, \text{tail}_{(j,i)}$.

cover without 2-cycles but with half-edges $\tilde{C}$ of the same weight. If a perfect matching M of G' matches nodes $i_{\{i,j\}}$ and $j_{\{i,j\}}$ as shown in Figure 2(a), then for one of the edges $(i,j),(j,i)$, both of the corresponding half-edges are excluded from $\tilde{C}$ and for the other edge, either both of the corresponding half-edges, or neither of the corresponding half-edges will be in $\tilde{C}$. If nodes $i_{\{i,j\}}$ and $j_{\{i,j\}}$ are matched as in Figure 2(b), M must contain appropriately either $\{\text{out}_i, \text{tail}_{(i,j)}\}, \{\text{out}_j, \text{tail}_{(j,i)}\}$ or $\{\text{in}_i, \text{head}_{(j,i)}\}, (\text{in}_j, \text{head}_{(i,j)})$, i.e., $\tilde{C}$ contains either $(i, v_{(i,j)})$ and $(j, v_{(j,i)})$ or $(v_{(j,i)}, i)$ and $(v_{(i,j)}, j)$. So indeed, condition (ii) of Definition 1 is satisfied. ◀

▶ **Lemma 3.** *Given a graph G, let $\tilde{C}$ be a cycle cover without 2-cycles but with half-edges. Then, we can construct three sets of node disjoint paths $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ in G, with total weight at least twice the weight of $\tilde{C}$.*

Proof. We use $\tilde{C}$ to construct a set F of directed and undirected edges with endpoints in V, and we then decompose F into three paths. For a pair of vertices i, j, if both $(i, v_{(i,j)})$ and $(v_{(i,j)}, j)$ are in $\tilde{C}$, then we replace it by the edge (i,j). If both $(i, v_{(i,j)})$ and $(j, v_{(j,i)})$, or $(v_{(i,j)}, j)$ and $(v_{(j,i)}, i)$ are in $\tilde{C}$, then we add undirected edge $\{i,j\}$. We think of an undirected edge as having two tails (and no heads) in the first case, and two heads (and no tails) in the second case. Note that it is then the case that each node is the head of one edge and the tail of one edge in F, by property (i) of Definition 1.

We will show how to find three sets of node-disjoint paths, such that each edge $(i,j) \in F$ is in exactly two of the paths, and for an undirected edge $\{i,j\} \in F$, there is one path that contains (i,j) and one path that contains (j,i). Note that the sum of the weights of these three sets of paths is exactly equal to twice the weight of $\tilde{C}$.

We consider a weakly connected component in (V,F). Note that a component has at least three nodes by property (ii) of Definition 1. If the component has no undirected edges, then it is a directed cycle, since each node is the head of one edge and the tail of another edge. Let F' be the edges in the component. We take two adjacent edges e_1, e_2 and make this the first path, the second path is $F' \backslash \{e_1\}$ and the third path is $F' \backslash \{e_2\}$.

Otherwise, if F' does contain undirected edges, note that (V,F') is a cycle if we ignore the direction of all edges. Moreover, it is the case that the number of undirected edges in the cycle is even and an undirected edge having two heads (resp. two tails) is followed on the cycle by an undirected edge having two tails (resp. two heads). To see this, recall that each

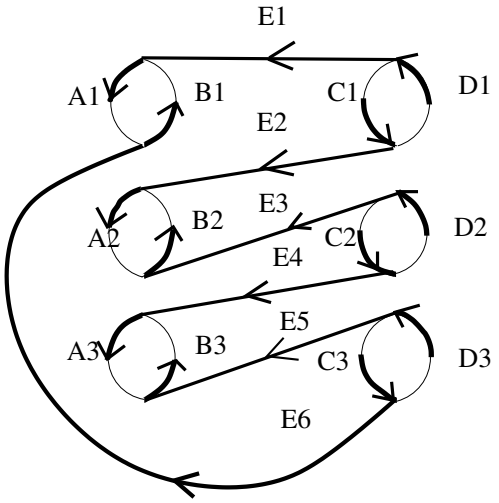


Figure 3 $E_1, E_2, \ldots, E_6$ represent directed paths. To P_1 we add edges on paths E_1, E_3, E_5 as well as edges $A_1, D_1, B_2, D_2, B_3, D_3$. To P_2 we add edges on paths E_2, E_4, E_6 as well as edges $B_1, C_1, A_2, C_2, A_3, C_3$. To P_3 we add edges on paths $E_1, E_2, \ldots, E_6$.

node is the head and the tail of one edge, and that undirected edges have either two heads or two tails.

The paths to be added to $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_3$ are now constructed as follows: to $\mathcal{P}_1$ we add the directed edges that point in clockwise direction and we add the undirected edges, which we make directed by directing them in clockwise direction. To $\mathcal{P}_2$ we add the directed edges that point in anticlockwise direction, and we add the undirected edges and direct them in anticlockwise direction. Finally, we also add all directed edges from F' to $\mathcal{P}_3$. An example of this procedure is shown in Figure 3.

There is one exception to the above construction. If the directed edges in F' form one path (possibly an empty path), then at least one of the two sets $\mathcal{P}_1, \mathcal{P}_2$ contains a cycle. In this case, we take one undirected edge in F' and reverse its direction in both $\mathcal{P}_1$ and $\mathcal{P}_2$.

Clearly, the paths are node disjoint, each directed edge is contained in two of the three sets $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, and for each undirected edge $\{i, j\}$, there is one set of paths that contains (i, j) and one set of paths that contains (j, i). ◀

▶ **Corollary 4.** *There exists a $\frac{2}{3}$-approximation algorithm for max ATSP.*

Proof. The optimal tour gives a set $\tilde{C}$ that is a cycle cover without 2-cycles but with half-edges, if for each edge (i, j) in the optimal tour, we include $(i, v_{(i,j)})$ and $(v_{(i,j)}, j)$ in $\tilde{C}$. Hence the maximum weight set $\tilde{C}$ has weight at least OPT, and it can be computed in polynomial time using Lemma 2. Using Lemma 3, we can thus find three sets of node disjoint paths, $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, of total weight $2OPT$. We can arbitrarily connect the paths in $\mathcal{P}_i$ into a tour without decreasing the weight, for $i = 1, 2, 3$ and hence, one of these tours has weight at least $\frac{2}{3}OPT$. ◀

Acknowledgements

The authors thank Marcin Mucha for useful comments on an earlier version of this paper.

References

1 Markus Bläser. An 8/13-approximation algorithm for the asymmetric maximum TSP. *J. Algorithms*, 50(1):23–48, 2004.

2 Markus Bläser and Bodo Manthey. Two approximation algorithms for 3-cycle covers. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization*, volume 2462 of *Lecture Notes in Computer Science*, pages 40–50. Springer, 2002.

3 Dany Breslauer, Tao Jiang, and Zhigen Jiang. Rotations of periodic strings and short superstrings. *J. Algorithms*, 24(2):340–353, 1997.

4 M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for finding a maximum weight Hamiltonian circuit. *Oper. Res.*, 27(4):799–809, 1979.

5 Haim Kaplan, Moshe Lewenstein, Nira Shafrir, and Maxim Sviridenko. Approximation algorithms for asymmetric tsp by decomposing directed regular multigraphs. *J. ACM*, 52(4):602–626, 2005. Preliminary version appeared in FOCS'03.

6 S. Rao Kosaraju, James K. Park, and Clifford Stein. Long tours and short superstrings (preliminary version). In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 166–177, 1994.

7 Lukasz Kowalik and Marcin Mucha. Deterministic 7/8-approximation for the metric maximum tsp. *Theor. Comput. Sci.*, 410(47-49):5000–5009, 2009.

8 Lukasz Kowalik and Marcin Mucha. 35/44-approximation for asymmetric maximum tsp with triangle inequality. *Algorithmica*, 59(2):240–255, 2011.

9 Moshe Lewenstein and Maxim Sviridenko. A 5/8 approximation algorithm for the maximum asymmetric tsp. *SIAM J. Discrete Math.*, 17(2):237–248, 2003.

10 Katarzyna E. Paluch, Marcin Mucha, and Aleksander Madry. A 7/9 - approximation algorithm for the maximum traveling salesman problem. In *APPROX-RANDOM*, pages 298–311, 2009.

11 Christos H. Papadimitriou and Mihalis Yannakakis. The traveling salesman problem with distances one and two. *Mathematics of Operations Research*, 18:1–11, 1993.

12 Jorma Tarhio and Esko Ukkonen. A greedy approximation algorithm for constructing shortest common superstrings. *Theor. Comput. Sci.*, 57:131–145, 1988.

Stabilization of Branching Queueing Networks*

Tomáš Brázdil[1] **and Stefan Kiefer**[2]

1 **Faculty of Informatics, Masaryk University, Czech Republic**
brazdil@fi.muni.cz
2 **Department of Computer Science, University of Oxford, United Kingdom**
stefan.kiefer@cs.ox.ac.uk

Abstract

Queueing networks are gaining attraction for the performance analysis of parallel computer systems. A Jackson network is a set of interconnected servers, where the completion of a job at server i may result in the creation of a new job for server j. We propose to extend Jackson networks by "branching" and by "control" features. Both extensions are new and substantially expand the modelling power of Jackson networks. On the other hand, the extensions raise computational questions, particularly concerning the stability of the networks, i.e, the ergodicity of the underlying Markov chain. We show for our extended model that it is decidable in polynomial time if there exists a controller that achieves stability. Moreover, if such a controller exists, one can efficiently compute a static randomized controller which stabilizes the network in a very strong sense; in particular, all moments of the queue sizes are finite.

1998 ACM Subject Classification G.3 Probability and Statistics

Keywords and phrases continuous-time Markov decision processes, infinite-state systems, performance analysis

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.507

1 Introduction

Queueing theory plays a central role in the performance analysis of computer systems. In particular, *queueing networks* are gaining attraction as models of parallel systems. A queueing network is a set of processing units (called *servers*), each of which performs tasks (called *jobs*) of a certain type. Each server has its own queue of jobs waiting to be processed. The successful completion of a job may trigger one (or more) new jobs (of possibly different type) that need to be processed as well. In addition to this "internal" job creation, so-called *open* queueing networks allow for new jobs to arrive "externally", i.e., from outside.

Queueing networks are a popular model for both hardware and software systems because of their simplicity and generality. On the hardware side, queueing networks can, e.g., be used for modeling multi-core processors, see e.g. [28] and the references in [8]. One advantage of queueing-based analyses is their scalability with growing parallelism; e.g., it is said in [20]: "Cycle-accurate full-system performance simulators do not scale well beyond a few tens of processor cores at best. As such, analytical models based on the theory of queueing systems, are a logical choice for developing a basic understanding of the fundamental tradeoffs in future, large-scale multi-core systems." On the software side, queueing networks are used for modeling message passing. It is said in [27]: "Two natural classes of systems can be modeled

* Tomáš Brázdil is supported by the Czech Science Foundation, grant No. P202/10/1469. Stefan Kiefer is supported by a DAAD postdoctoral fellowship.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 507–518

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

using such a framework: asynchronous programs on a multi-core computer and distributed programs communicating on a network." Of course, the realm of queueing networks stretches far beyond computer science, see [3, 7].

The simplest queueing networks are so-called *Jackson networks* [15]: Given two servers $i, j \in \{1, \dots, n\}$, there is a "rule" of the form $i \xhookrightarrow{p_{ij}} j$ which specifies the probability p_{ij} that the completion of an i-job results in the creation of a j-job. There are also rules $i \xhookrightarrow{p_{i0}} \varepsilon$ where $p_{i0} = 1 - \sum_j p_{ij}$ specifies the probability that no new job is created. Each server i has a *rate* μ_i with which an i-job is processed if there is one. In addition, there is a rate α_i with which i-jobs arrive from outside the network. The processing times and the external arrivals are exponentially distributed, so that a Jackson network describes a *continuous-time Markov chain (CTMC)*. It was shown in Jackson's paper [15] that if the rate λ_i of internal *and* external arrivals at server i is less than μ_i for all i, then the network is *stable*, i.e., the average queue length is finite and almost surely all queues are empty infinitely often. Moreover, Jackson networks allow a *product form*, i.e., the steady-state distribution of the queue lengths can be expressed as a product of functions $\pi_i(k)$, where $\pi_i(k)$ is the steady-state probability that queue i has length k.

▶ **Example 1** (network processor)**.** In [1], Jackson networks are used to model network processors, i.e., chips that are specifically targeted at networking applications—think of a router. We describe the model from [1] (sections 4.1 and 4.2, slightly adapted). Before packets are processed in the "master processor" M, they pass through the "data plane" D, from which a fraction q of packets needs to be processed first in the "control plane" C:

$$D \xhookrightarrow{1-q} M \qquad D \xhookrightarrow{q} C \qquad C \xhookrightarrow{1} M \qquad M \xhookrightarrow{1} \varepsilon$$

An "arrival manager" A sends some packets (fraction d_0) directly to D, but others (fractions $d_1, \dots, d_n$ with $d_0 + d_1 + \cdots + d_n = 1$) are sent to "slave processors" $S_1, \dots, S_n$ to assist the master. Some packets (fraction b) still need the attention of the master after having been processed by a slave:

$$A \xhookrightarrow{d_0} D \qquad A \xhookrightarrow{d_i} S_i \qquad S_i \xhookrightarrow{b} D \qquad S_i \xhookrightarrow{1-b} \varepsilon \quad , \quad i \in \{1, \dots, n\}.$$

Jackson networks and their extensions have been thoroughly studied, but they are restricted in their modelling capabilities, as (i) the completion of a job may trigger at most one job, and (ii) there is no nondeterminism that would allow to control the output probabilities of a server. Considering (i), it seems unnatural to assume that a distributed program communicating on a network produces at most one message at the end of its computation. Considering (ii), the "arrival manager" A in Example 1 may want to flexibly pass incoming packets to the master or one of the slaves, possibly depending on the current load. These restrictions have not been fully addressed, not even in isolation. In this paper we introduce *controlled branching queueing networks*, which are Jackson-like networks but allow for both nondeterminism ("controlled") and the creation of more than one job ("branching").

Both extensions directly raise computational issues. We show in Example 2 on page 511 that even purely stochastic branching networks do not allow a product form, which illustrates the mathematical challenge[1] of this extension and poses the question for an effective criterion that allows to determine whether the network is stable, i.e., returns to the empty state

[1] It is noted in [14] that "[...] virtually all of the models that have been successfully analyzed in classical queueing network theory are models having a so-called product form stationary distribution."

infinitely often. Moreover, due to the nondeterminism, we now deal with *continuous-time Markov decision processes (CTMDPs)*. Our main theorem (Theorem 3) states that if there exists any scheduler resolving the nondeterminism in such a way that the controlled branching network is stable, then there exists a *randomized static* scheduler that achieves stability as well, where by "randomized static" we mean that the decisions may be randomized but not dependent on the current state (such as the load) of the system. Moreover, the existence of such a stabilizing scheduler and the scheduler itself can be determined in polynomial time, and, finally, the randomized static scheduler is stabilizing in a very strong sense, in particular, all moments of the queue sizes are finite.

Related work. We use nondeterminism to describe systems whose behaviour is not completely specified. A system *designer* can then resolve the nondeterminism to achieve certain goals, in our case stability. Although nondeterminism is a very well established modelling feature of probabilistic systems (see e.g. [19]), the literature on automatic design of stabilizing controllers for queueing networks is sparse. *Flow-controlled* networks [26, 21] allow to control only the external arrival stream or the service rates (see also [2] and the references therein). The authors of [18, 13] consider queueing networks with fewer servers than job types, so that the controller needs to assign servers to queues. As in [18, 13], we also use linear programming to design a controller, but our aim is different: we allow the controller to influence the production of the individual queues, and we study the complexity of designing stabilizing controllers and the nature of such controllers. There has been a substantial amount of work in the last years analyzing probabilistic systems with "branching features", most prominently on *recursive Markov chains* [12, 11] and *probabilistic pushdown systems* [10, 5]. While these models allow for a probabilistic splitting of tasks by pushing new procedures on a stack, the produced tasks are processed in a strictly sequential manner, whereas the queues in a queueing network process jobs in parallel and in continuous time. Recently, *probabilistic split-join systems* were introduced [16], which allow for branching but not for external arrivals, and assume unlimited parallelism. In [17, chapter 8] a queueing model with multiple classes of tasks and "feedback" is discussed, which is similar to our branching except that there is only one server, hence there is no parallelism. Algorithmic theory of queueing systems has also attracted some attention in the past. In particular, for *closed* (i.e., without external arrivals) queueing systems, [24] shows EXP-completeness of minimizing a weighted throughput of the queues.

2 Preliminaries

Numbers. We use $\mathbb{Z}, \mathbb{Q}, \mathbb{R}$ for the sets of integer, rational, real numbers, respectively, and $\mathbb{N}, \mathbb{Q}_{\geq 0}, \mathbb{R}_{\geq 0}$ for their respective subsets of nonnegative numbers.

Vectors and Matrices. Let $n \geq 1$. We use boldface letters for denoting vectors $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \mathbb{R}^n$. Vectors are row vectors per default, and we use superscript ^{T} for transpose, so that $\boldsymbol{x}^T$ denotes a column vector. If the dimension n is clear from the context, we write $\mathbf{0} := (0, \ldots, 0)$, $\mathbf{1} := (1, \ldots, 1)$, and $\boldsymbol{e}^{(i)} = (0, \ldots, 0, 1, 0, \ldots, 0)$ for the vector with the 1 at the ith component ($1 \leq i \leq n$). It is convenient to define $\boldsymbol{e}^{(0)} := \mathbf{0}$. For two vectors $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ we write $\boldsymbol{x} \sim \boldsymbol{y}$ with $\sim \in \{=, <, \leq, >, \geq\}$ if the respective relation holds componentwise. For a vector $\boldsymbol{x} \in \mathbb{R}^n$ we denote its *1-norm* by $\|\boldsymbol{x}\| := \sum_{i=1}^{n} |\boldsymbol{x}_i|$. When $\boldsymbol{x} \in \mathbb{N}^n$ is a vector of queue sizes, we refer to $\|\boldsymbol{x}\|$ as the *total queue size*. For a matrix $A \in \mathbb{R}^{n \times n}$, we write A_i for its ith row, i.e., $A_i = (A_{i1}, \ldots, A_{in})$.

CTMDP. A *continuous-time Markov decision process (CTMDP)* consists of an at most countable set S of states, an initial state $s_1 \in S$, a set of actions Σ [2], and a transition rate $q(s, \sigma, s') \geq 0$ for each pair of states $s, s' \in S$ and each action $\sigma \in \Sigma$ (here $q(s, \sigma, s') = 0$ means that the transition from s to s' never occurs). We define a *continuous-time Markov chain (CTMC)* to be a CTMDP whose set of actions Σ is a singleton (we usually do not write the only action explicitly, so the transition rates of a CTMC are denoted by $q(s, s')$, etc.).

Intuitively, a run of a CTMDP starts in s_1 and then evolves in so-called epochs. Assume that (after the previous epoch) the system is in a state s. The next epoch consists of the following phases: First, a scheduler chooses an action $\sigma \in \Sigma$ to be executed. Second, a waiting time for transition to each state $s' \in S$ is chosen according to the exponential distribution with the rate $q(s, \sigma, s')$ (here we assume that if $q(s, \sigma, s') = 0$, then the waiting time is ∞). The transitions compete in a way such that the one with the least waiting time is executed and the state of the CTMDP is chosen accordingly (the other transitions are subsequently discarded).

Formally, a *run* is an infinite sequence $s_1, \sigma_1, t_1, s_2, \sigma_2, t_2, \ldots \in (S \times \Sigma \times \mathbb{R}_{\geq 0})^\omega$. We denote by *Run* the set of all runs. A *scheduler* is a function Θ which assigns to every finite path $s_1, \sigma_1, t_1, s_2, \sigma_2, t_2, \ldots s_n \in (S \times \Sigma \times \mathbb{R}_{\geq 0})^* \times S$ a probability distribution $\Theta(w)$ on actions (i.e. $\Theta(w) : \Sigma \to [0, 1]$ satisfies $\sum_{\sigma \in \Sigma} \Theta(w)(\sigma) = 1$). For technical reasons, we have to restrict ourselves to measurable schedulers (for details see e.g. [23]).

We work with a measurable space of runs $(\mathit{Run}, \mathcal{F})$ where $\mathcal{F}$ is the smallest σ-algebra generated by basic cylinders (i.e. sets of runs with common finite prefix) in a standard way. Every scheduler Θ induces a unique probability measure $\Pr_\Theta$ on $\mathcal{F}$ determined by the probabilities of the basic cylinders. For detailed definitions see [23]. Then each scheduler Θ *induces* a stochastic process $(x(t) \mid t \in \mathbb{R}_{\geq 0})$ on the probability space $(\mathit{Run}, \mathcal{F}, \Pr_\Theta)$ where $x(t)$ is the current state of the run in time t, i.e., each $x(t)$ is a random variable defined by

$$x(t)(s_1, \sigma_1, t_1, s_2, \sigma_2, t_2, \ldots) = s_i \quad , \quad \sum_{j=1}^{i-1} t_i \leq t \text{ and } \sum_{j=1}^{i} t_i \geq t \,.$$

A scheduler Θ is *memoryless* if for every path $w = s_1, \sigma_1, t_1, s_2, \sigma_2, t_2, \ldots s_{n+1} \in (S \times \Sigma \times \mathbb{R}_{\geq 0})^* \times S$ we have that $\Theta(w) = \Theta(s_{n+1})$.

Networks. Define $\mathsf{R}^{(n,K)} := \{\boldsymbol{r} \in \mathbb{N}^n \mid \boldsymbol{r}_1 + \cdots + \boldsymbol{r}_n \leq K\}$. A *production function* for (n, K) is a function $\mathit{Prob} : R \to \mathbb{Q} \cap (0, 1]$ with $R \subseteq \mathsf{R}^{(n,K)}$ such that $\sum_{\boldsymbol{r} \in R} \mathit{Prob}(\boldsymbol{r}) = 1$. A *controlled branching network with n queues and branching factor K* consists of an arrival rate $\mu_0 \in \mathbb{Q} \cap (0, \infty)$, queue rates $\mu_i \in \mathbb{Q} \cap (0, \infty)$ for $i \in \{1, \ldots, n\}$, an arrival production function $\mathit{Prob}_0 : R_0 \to \mathbb{Q} \cap (0, 1]$ for (n, K), and finite action sets $\Sigma_1, \ldots, \Sigma_n$ as follows. An action $\sigma_i \in \Sigma_i$ assigns to queue i a production function $\mathit{Prob}_i(\sigma_i) : R_i(\sigma_i) \to \mathbb{Q} \cap (0, 1]$ for (n, K). Define $\Sigma := \Sigma_1 \times \cdots \times \Sigma_n$. If $\sigma = (\sigma_1, \ldots, \sigma_n) \in \Sigma$, we write $R_i(\sigma)$, $\mathit{Prob}_i(\sigma)$ and $R_0(\sigma)$, $\mathit{Prob}_0(\sigma)$ to mean $R_i(\sigma_i)$, $\mathit{Prob}_i(\sigma_i)$ and R_0, Prob_0. Observe that the rates μ_i do not depend on actions. This simplification is without loss of generality.[3] We assume a nonzero

[2] Usually, each state has its own set of available actions. As this feature is not needed for queueing networks, we stick to the simpler version in which all actions are always available.

[3] To show that this assumption is w.l.o.g. one can employ the standard "uniformization" trick. More precisely, assume that the actions of a queue i have different rates. Define μ_i to be the maximum of all rates of Σ_i and compensate by "adding self-loops", i.e., make the actions of Σ_i generate a new job for queue i with a suitable probability. This effectively substitutes a transition with longer delay by possibly several transitions with delay μ_i. As static schedulers can be easily translated between the original and the transformed system, our results remain valid.

arrival stream, i.e., there is $\boldsymbol{r} \in R_0$ with $\boldsymbol{r} \neq \mathbf{0}$. We define the *size* of a controlled network by $n + K + (\sum_{i=0}^{n} |\mu_i|) + |R_0| + |Prob_0| + \sum_{i=1}^{n} \sum_{\sigma_i \in \Sigma_i} |R_i(\sigma_i)| + |Prob_i(\sigma_i)|$, where $|\mu_i|$ etc. means the description size assuming the rationals are represented as fractions of integers in binary. A controlled branching network induces a CTMDP with state space $\mathbb{N}^n$ (the queue sizes), initial state $\mathbf{0}$, action set Σ, and transition rates

$$q(\boldsymbol{x}, \sigma, \boldsymbol{y}) = \sum_{i \in \{0,1,\ldots,n\} : i=0 \vee \boldsymbol{x}_i \neq 0} \; \sum_{\boldsymbol{r} \in R_i(\sigma) : \boldsymbol{y} = \boldsymbol{x} - \boldsymbol{e}^{(i)} + \boldsymbol{r}} \mu_i Prob_i(\sigma)(\boldsymbol{r}) \qquad \text{for } \boldsymbol{x}, \boldsymbol{y} \in \mathbb{N}^n,\ \sigma \in \Sigma.$$

Interpreting this definition, there is a "race" between external arrivals (rate μ_0) and the nonempty queues (rates μ_i); if the external arrivals win, new jobs are added according to $Prob_0(\sigma)$; if queue i wins, one i-job is removed and new jobs are added according to $Prob_i(\sigma)$.

An *purely stochastic branching network* is a controlled branching network with $\Sigma = \{\sigma\}$, i.e., with a unique action for each queue. Hence, the induced CTMDP is a CTMC. In the purely stochastic case we write only R_i for $R_i(\sigma)$ etc. If $Prob_i(\boldsymbol{r}) = p$ in the purely stochastic case, we use in examples the notation $i \overset{p}{\hookrightarrow} \boldsymbol{r}$, where we often write $\boldsymbol{r} \in \mathsf{R}^{(n,K)}$ as a multiset without curly brackets. For instance, if $n = 2$, we write $1 \overset{p}{\hookrightarrow} 1, 2$ and $1 \overset{p}{\hookrightarrow} 2, 2$ and $1 \overset{p}{\hookrightarrow} \varepsilon$ to mean $Prob_1(\boldsymbol{r}) = p$ with $\boldsymbol{r} = (1, 1)$ and $\boldsymbol{r} = (0, 2)$ and $\boldsymbol{r} = (0, 0)$, respectively.

Fixing a controlled network $\mathcal{N}$ and a scheduler Θ for the CTMDP induced by $\mathcal{N}$, we obtain a stochastic process $\mathcal{N}_\Theta = (\boldsymbol{x}(t) \mid t \in \mathbb{R}_{\geq 0})$, where $\boldsymbol{x}(0) = \mathbf{0} \in \mathbb{N}^n$, which evolves according to the dynamics of $\mathcal{N}$ and the scheduler Θ. In the purely stochastic case we drop the subscript Θ, and so we identify a network $\mathcal{N}$ with its induced stochastic process.

▶ **Example 2** (no product form)**.** Consider the purely stochastic branching network with $0 \overset{1}{\hookrightarrow} 1, 2$ and $1 \overset{1}{\hookrightarrow} \varepsilon$ and $2 \overset{1}{\hookrightarrow} \varepsilon$. If its stationary distribution π (for a definition of stationary distribution see before Theorem 3) had product from, the queues would be "independent in steady-state", i.e., $\pi(\boldsymbol{x}_2 \geq 1 \mid \boldsymbol{x}_1 \geq 1) = \pi(\boldsymbol{x}_2 \geq 1)$, where by $\boldsymbol{x}$ we mean $\boldsymbol{x}(t)$ in steady state. However, if μ_0 is much smaller than $\mu_1 = \mu_2$, then we have $\pi(\boldsymbol{x}_2 \geq 1 \mid \boldsymbol{x}_1 \geq 1) > \pi(\boldsymbol{x}_2 \geq 1)$, intuitively because $\boldsymbol{x}_1 \geq 1$ probably means that there was an arrival recently, so that $\boldsymbol{x}_2 \geq 1$ is more likely than usual. More concretely, let $\mu_0 = 1$ and $\mu_1 = \mu_2 = 3$ and consider the 2-state Markov chain obtained by assuming that each arrival leads to the state $(1, 1)$ and each completion of any job leads to the state $(0, 0)$. By computing the stationary distribution π' of this 2-state Markov chain in the standard way, we obtain $\pi'((0, 0)) = 6/7$ and $\pi'((1, 1)) = 1/7$. Since this 2-state Markov chain "underapproximates" the CTMC induced by the network, we have $\pi(\boldsymbol{x}_1 \geq 1 \ \wedge \ \boldsymbol{x}_2 \geq 1) \geq 1/7$. On the other hand, by considering the two queues separately, the standard formula for the M/M/1 queue gives $\pi(\boldsymbol{x}_1 \geq 1) = \pi(\boldsymbol{x}_2 \geq 1) = 1/3$. Product form would imply $\pi(\boldsymbol{x}_1 \geq 1 \wedge \boldsymbol{x}_2 \geq 1) = \pi(\boldsymbol{x}_1 \geq 1) \cdot \pi(\boldsymbol{x}_2 \geq 1) = 1/9$, contradicting the inequality above.

3 Results

We focus on the stability of purely stochastic and controlled branching networks. Our notion of stability requires that the network is completely empty infinitely many times. Given a stochastic process $(\boldsymbol{x}(t) \mid t \in \mathbb{R}_{\geq 0})$, we say that the process is *ergodic* if the expected return time to $\mathbf{0}$ is finite. More formally, define a random variable R by

$$R \quad := \quad \inf_{t>0} \{t \mid \boldsymbol{x}(t) = \mathbf{0}, \exists t' < t : \boldsymbol{x}(t') \neq \mathbf{0}\} \ .$$

Then the process is ergodic iff $\mathbb{E}[R] < \infty$. In the controlled case, we say that a scheduler Θ for $\mathcal{N}$ is *ergodic for* $\mathcal{N}$ if $\mathcal{N}_\Theta$ is ergodic. In the following we use stability and ergodicity

interchangeably. A scheduler Θ is *static* if it always chooses the same fixed distribution on actions. Note that static schedulers are memoryless. If in a stochastic process $(\boldsymbol{x}(t) \mid t \in \mathbb{R}_{\geq 0})$ the limit $\pi(\boldsymbol{x}) := \lim_{t\to\infty} \Pr(\boldsymbol{x}(t) = \boldsymbol{x})$ exists for all $\boldsymbol{x} \in \mathbb{N}^n$ and $\sum_{\boldsymbol{x}\in\mathbb{N}^n} \pi(\boldsymbol{x}) = 1$, then $\pi : \mathbb{R}_{\geq 0} \to [0, 1]$ is called the *stationary distribution*.

▶ **Theorem 3.** *Let $\mathcal{N}$ be a controlled branching network. It is decidable in polynomial time whether there exists an (arbitrary) ergodic scheduler for $\mathcal{N}$. If it exists, one can compute, in polynomial time, a* static randomized *ergodic scheduler for $\mathcal{N}$ with stationary distribution π such that there exists an exponential moment of the total number of waiting jobs, i.e., there is $\delta > 0$ such that $\sum_{\boldsymbol{x}\in\mathbb{N}^n} \exp(\delta \|\boldsymbol{x}\|)\pi(\boldsymbol{x})$ exists.*

To prove Theorem 3 we generalize the concept of *traffic equations* (see e.g. [6]) from the theory of Jackson networks. Intuitively, the traffic equations express the fact that the inflow of jobs to a given queue must be equal to the outflow. Remarkably, the traffic equations characterize the stability of the Jackson network. More precisely, a Jackson network is stable if and only if there is a solution of the traffic equations whose components are strictly smaller than the rates of the corresponding queues (we call such a solution *deficient*).

We show how to extend the traffic equations so that they characterize the stability of controlled branching networks. For a smooth presentation, we start with *purely stochastic* branching networks and add control later on. Hence, the overall plan of the proof of Theorem 3 is as follows: Set up traffic equations for purely stochastic branching networks and show that if there is a deficient solution of these equations, then the network is stable. This result, presented in Section 3.1 (Proposition 4), is of independent interest and requires the construction of a suitable *Lyapunov function*. Then, in Section 3.2, we generalize the traffic equations to controlled branching networks and show that any ergodic scheduler determines a deficient solution (Proposition 10). This solution naturally induces a static scheduler, which, when fixed, determines an purely stochastic network with deficiently solvable traffic equations. Propositions 4 and 10 imply Theorem 3 and provide some additional results.

3.1 Purely stochastic branching networks

Assume that $\mathcal{N}$ is purely stochastic, i.e., there is a single action for each queue. In such a case the CTMDP induced by the network is in fact a CTMC. We associate the following quantities to a network, which will turn out to be crucial for its performance. Let $\boldsymbol{\mu} := (\mu_1, \ldots, \mu_n)$. Let $\boldsymbol{\alpha} \in \mathbb{R}^n_{\geq 0}$ be the vector with $\boldsymbol{\alpha}_i := \mu_0 \sum_{\boldsymbol{r}\in R_0} \mathit{Prob}_0(\boldsymbol{r})\boldsymbol{r}_i$; i.e., $\boldsymbol{\alpha}_i$ indicates the expected number of external arrivals at queue i per time unit. Note that $\boldsymbol{\alpha} \neq \mathbf{0}$, as we assume a nonzero arrival stream. Let $A \in \mathbb{R}^{n\times n}_{\geq 0}$ be the matrix with $A_{ij} := \sum_{\boldsymbol{r}\in R_i} \mathit{Prob}_i(\boldsymbol{r})\boldsymbol{r}_j$; i.e., A_{ij} indicates the expected production of j-jobs when queue i fires. W.l.o.g. we assume that all queues are "reachable", i.e., for all queues i there is $j \in \mathbb{N}$ with $(\boldsymbol{\alpha}A^j)_i \neq 0$. We define a set of *traffic equations*

$$\lambda_j \quad = \quad \alpha_j + \sum_{i=1}^{n} \lambda_i \cdot A_{ij} \quad , \quad j \in \{1, \ldots, n\}, \tag{1}$$

in matrix form:

$$\boldsymbol{\lambda} = \boldsymbol{\alpha} + \boldsymbol{\lambda}A\,. \tag{2}$$

We prove the following proposition.

▶ **Proposition 4.** *Assume that* $\boldsymbol{\lambda} \in \mathbb{R}^n_{\geq 0}$ *solves the traffic equations* (2) *and satisfies* $\boldsymbol{\lambda} < \boldsymbol{\mu}$. *Then the following conclusions hold:*

1. *The process* $\mathcal{N}$ *is ergodic, i.e., the expected return time to* $\mathbf{0}$ *is finite.*
2. *There exists a stationary distribution* π *such that there exists an exponential moment of the total queue size, i.e., there is* $\delta > 0$ *such that* $\sum_{\boldsymbol{x} \in \mathbb{N}^n} \exp(\delta \, \|\boldsymbol{x}\|)\pi(\boldsymbol{x})$ *exists.*

The key step to the proof of Proposition 4 is to construct a so-called *Lyapunov function* with respect to which the process $\mathcal{N}$ exhibits a "negative drift". This is in fact a classical technique for showing the stability of queueing systems [22]; the difficulty lies in finding a suitable Lyapunov function. The "drift" of $\mathcal{N}$ is given by the *mean velocity vector* $\boldsymbol{\Delta}(\boldsymbol{x}) \in \mathbb{R}^n_{\geq 0}$ of $\mathcal{N}$, defined by $\boldsymbol{\Delta}(\boldsymbol{x}) := \lim_{h \to 0^+} \mathbb{E}\left[\boldsymbol{x}(t+h) - \boldsymbol{x}(t) \mid \boldsymbol{x}(t) = \boldsymbol{x}\right]/h$. The limit exists, is independent of t, and we have

$$\boldsymbol{\Delta}(\boldsymbol{x}) = \boldsymbol{\alpha} + \sum_{i : \boldsymbol{x}_i \neq 0} \mu_i(-\boldsymbol{e}^{(i)} + A_i)\,. \tag{3}$$

The following lemma is implicitly proved in [9, theorem 1].

▶ **Lemma 5.** *Suppose that a function* $\widetilde{V} : \mathbb{R}^n_{\geq 0} \to \mathbb{R}_{\geq 0}$ *is two times continuously differentiable,* $\widetilde{V}(\boldsymbol{x}) = 0$ *implies* $\boldsymbol{x} = \mathbf{0}$, *and that there is* $\gamma > 0$ *such that we have*

$$\boldsymbol{\Delta}(\boldsymbol{x})\big(\widetilde{V}'(\boldsymbol{x})\big)^T \leq -\gamma \qquad \text{for all } \boldsymbol{x} \neq \mathbf{0},$$

where $\widetilde{V}'(\boldsymbol{x})$ *denotes the gradient of* $\widetilde{V}$ *at* $\boldsymbol{x}$. *Then the conclusions of Proposition 4 holds.*

Following [9], we construct the Lyapunov function $\widetilde{V}$ in two stages: we first define a suitable *piecewise linear* function $V : \mathbb{N}^n \to \mathbb{R}_{\geq 0}$; then V is *smoothed* to obtain $\widetilde{V}$. For the definition of V we need the following lemma.

▶ **Lemma 6.** *The matrix series* $A^* := \sum_{i=0}^{\infty} A^i$ *converges ("exists") in* $\mathbb{R}^{n \times n}_{\geq 0}$ *and is equal to* $(I - A)^{-1}$.

Define vectors $\boldsymbol{q}^{(1)}, \ldots, \boldsymbol{q}^{(n)} \in \mathbb{R}^n_{\geq 0}$ by setting $\boldsymbol{q}^{(i)T} := \boldsymbol{a}^{(i)T} / \left\|\boldsymbol{a}^{(i)}\right\|$, where $\boldsymbol{a}^{(i)T}$ is the ith column of A^*. Observe that we have $\mathbf{1}\boldsymbol{q}^{(i)T} = 1$ for all i. Define the function $V : \mathbb{R}^n_{\geq 0} \to \mathbb{R}_{\geq 0}$ by $V(\boldsymbol{x}) := \max_i\{\boldsymbol{x}\boldsymbol{q}^{(i)T}\}$. We will use the following property of V:

▶ **Lemma 7.** *If* $\mathbf{0} \neq \boldsymbol{x} \in \mathbb{R}^n_{\geq 0}$ *and* $\boldsymbol{x}_i = 0$, *then* $\boldsymbol{x}\boldsymbol{q}^{(i)T} < V(\boldsymbol{x})$.

Lemma 7 is not obvious; in [4] we use Farkas' lemma for the proof. The following lemma describes the crucial "negative drift" property of V:

▶ **Lemma 8.** *There is* $\gamma > 0$ *such that we have*

$$\boldsymbol{\Delta}(\boldsymbol{x})\big(V'(\boldsymbol{x})\big)^T \leq -\gamma \qquad \text{for all } \boldsymbol{x} \neq \mathbf{0}$$

and all subgradient vectors $V'(\boldsymbol{x})$ *of* V *at* $\boldsymbol{x}$. *More precisely, one can choose*

$$\gamma := \min_i(\mu_i - \boldsymbol{\lambda}_i)/\left\|\boldsymbol{a}^{(i)}\right\|\,,$$

where $\boldsymbol{a}^{(i)T}$ *is the* i*th column of* A^*.

▶ **Example 9.** Consider the network with

$$0 \overset{1}{\hookrightarrow} 1 \qquad \begin{matrix} 1 \overset{1/5}{\longrightarrow} 2, 2 \\ 1 \overset{4/5}{\longrightarrow} \varepsilon \end{matrix} \qquad \begin{matrix} 2 \overset{1/6}{\longrightarrow} 1, 2 \\ 2 \overset{5/6}{\longrightarrow} \varepsilon \end{matrix}\,,$$

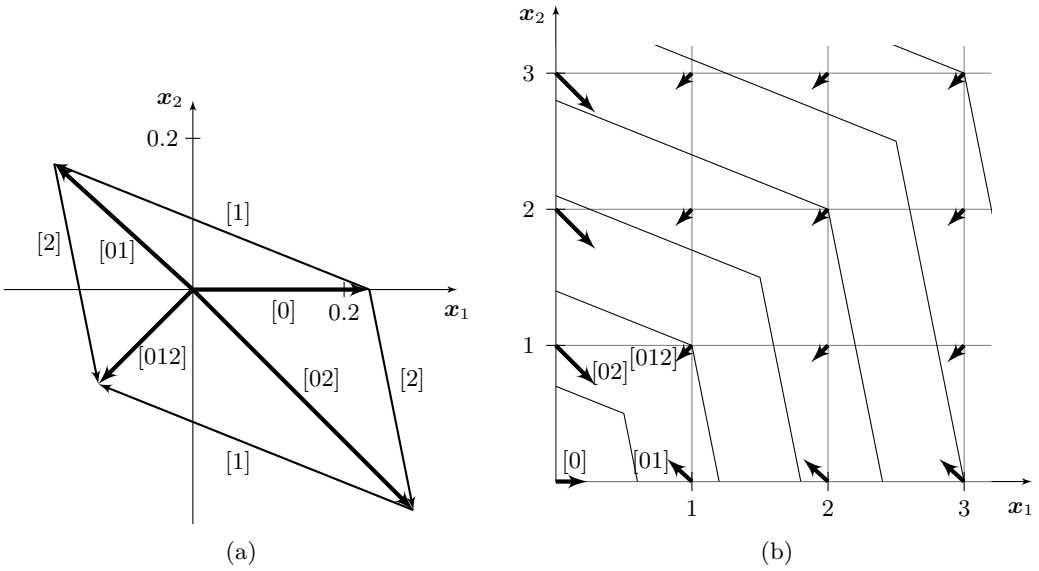


Figure 1 Illustration of negative drift.

arrival rate $\mu_0 = 7/30$, and $\boldsymbol{\mu} = (5/12, 7/20)$, so that $\mu_0 + \mu_1 + \mu_2 = 1$. Let us write $[0] := \boldsymbol{\alpha} = \mu_0(1,0)$, $[1] := \mu_1(-\boldsymbol{e}^{(1)} + A_1)$, $[2] := \mu_2(-\boldsymbol{e}^{(2)} + A_2)$, $[01] := [0] + [1]$, $[02] := [0] + [2]$, $[012] := [0] + [1] + [2]$. These vectors are shown in Figure 1 (a). The mean velocity vector $\boldsymbol{\Delta}(\boldsymbol{x})$ is one of the vectors $[0], [01], [02], [012]$, depending on which components of $\boldsymbol{x}$ are nonzero. The vector field in Figure 1 (b) shows the corresponding vectors for several $\boldsymbol{x} \in \mathbb{N}^2$. The connected line segments indicate points $\boldsymbol{x}$ with the same value of $V(\boldsymbol{x}) = \max\{\boldsymbol{x}\boldsymbol{q}^{(1)T}, \boldsymbol{x}\boldsymbol{q}^{(2)T}\} = \max\{\frac{5}{6}\boldsymbol{x}_1 + \frac{1}{6}\boldsymbol{x}_2, \frac{2}{7}\boldsymbol{x}_1 + \frac{5}{7}\boldsymbol{x}_2\}$ (values $0.5, 1, 1.5, \ldots$). It can be seen from the figure that the drift is negative with respect to the gradient of V, if $\boldsymbol{x} \neq \boldsymbol{0}$.

Proof of Lemma 8. Let $\boldsymbol{x} \neq \boldsymbol{0}$. We need to show $\boldsymbol{\Delta}(\boldsymbol{x})\boldsymbol{q}^{(i)T} \leq -\gamma$ for all i with $\boldsymbol{x}\boldsymbol{q}^{(i)T} = V(\boldsymbol{x})$. W.l.o.g. we assume that $\boldsymbol{x}\boldsymbol{q}^{(1)T} = V(\boldsymbol{x})$ and show only $\boldsymbol{\Delta}(\boldsymbol{x})\boldsymbol{q}^{(1)T} \leq -\gamma$. By Lemma 7 we have $\boldsymbol{x}_1 \neq 0$. It follows from the property $(I - A)A^* = I$ and the definition of $\boldsymbol{q}^{(1)}$ that we have

$$(-\boldsymbol{e}^{(1)} + A_1)\boldsymbol{q}^{(1)T} = -1/\left\|\boldsymbol{a}^{(1)}\right\| \qquad \text{and} \qquad (-\boldsymbol{e}^{(i)} + A_i)\boldsymbol{q}^{(1)T} = 0 \quad \text{for } 2 \leq i \leq n. \tag{4}$$

Hence we have:

$$\begin{aligned}
\boldsymbol{\Delta}(\boldsymbol{x})\boldsymbol{q}^{(1)T} &= \left(\boldsymbol{\alpha} + \sum_{i:\boldsymbol{x}_i \neq 0} \mu_i(-\boldsymbol{e}^{(i)} + A_i)\right)\boldsymbol{q}^{(1)T} && \text{by (3)} \\
&= \boldsymbol{\alpha}\boldsymbol{q}^{(1)T} - \mu_1/\left\|\boldsymbol{a}^{(1)}\right\| && \text{by (4) and } \boldsymbol{x}_1 \neq 0 \\
&\leq -\gamma + \boldsymbol{\alpha}\boldsymbol{q}^{(1)T} - \boldsymbol{\lambda}_1/\left\|\boldsymbol{a}^{(1)}\right\| && \text{by the definition of } \gamma \\
&= -\gamma + \left(\boldsymbol{\alpha} + \sum_{i=1}^{n} \boldsymbol{\lambda}_i(-\boldsymbol{e}^{(i)} + A_i)\right)\boldsymbol{q}^{(1)T} && \text{by (4)} \\
&= -\gamma + (\boldsymbol{\alpha} + \boldsymbol{\lambda}(-I + A))\,\boldsymbol{q}^{(1)T} \\
&= -\gamma + \boldsymbol{0}\boldsymbol{q}^{(1)T} = -\gamma && \text{by the traffic equation.}
\end{aligned}$$

◀

Using integration, one can obtain a two times continuously differentiable function $\widetilde{V}$ satisfying the conditions in Lemma 5: the function V is smoothed by defining $\widetilde{V}(\boldsymbol{x})$, for all $\boldsymbol{x}$, as an "average" of the values $V(\boldsymbol{y})$ where $\boldsymbol{y}$ belongs to a small ball around $\boldsymbol{x}$; see the appendix of [9] for the formal details. This concludes the proof of Proposition 4.

3.2 Controlled branching networks

In this subsection we generalize the traffic equations (2) to deal with an arbitrary controlled branching network $\mathcal{N}$. To obtain a distribution on actions for a static randomized ergodic scheduler, we assign variables to actions instead of queues, i.e., for every action ξ of the network we introduce a variable λ_ξ capturing the *rate of firing the action* ξ. Denote by $\bar{\Sigma}$ the set $\bigcup_{i=1}^{n} \Sigma_i$. Given $\zeta \in \bar{\Sigma}$ and $j \in \{1, \ldots, n\}$, we denote by $A_{\zeta j}$ the average number of jobs added to the queue j when the action ζ fires, i.e., for $\zeta \in \Sigma_i$ we set

$$A_{\zeta j} := \sum_{\boldsymbol{r} \in R_i(\zeta)} \mathit{Prob}_i(\zeta)(\boldsymbol{r}) \cdot \boldsymbol{r}_j \,.$$

We generalize (2) to the *traffic LP* presented in Figure 2, where the variable δ is intended to bound, for all j, the probability that queue j is busy.

min δ subject to

$$\begin{aligned}
\sum_{\xi \in \Sigma_j} \lambda_\xi &= \boldsymbol{\alpha}_j + \sum_{i=1}^{n} \sum_{\zeta \in \Sigma_i} \lambda_\zeta \cdot A_{\zeta j} && j \in \{1, \ldots, n\} \\
\delta &\geq \frac{\sum_{\xi \in \Sigma_j} \lambda_\xi}{\mu_j} && j \in \{1, \ldots, n\} \\
\lambda_\xi &\geq 0 && \xi \in \bar{\Sigma}
\end{aligned}$$

Figure 2 The traffic LP.

We prove the following

▶ **Proposition 10.**

1. *If there exists an arbitrary ergodic scheduler for $\mathcal{N}$, then the traffic LP can be solved with* $\min \delta < 1$.
2. *If the traffic LP is solved with* $\min \delta < 1$, *one can compute in polynomial time a static randomized ergodic scheduler Θ_s for $\mathcal{N}$. Moreover, denoting by ρ_i the* utilization $\lim_{t \to \infty} \Pr(\boldsymbol{x}_i(t) \neq 0)$ *of the queue i, the scheduler Θ_s minimizes* $\max_i \rho_i$ *among all memoryless ergodic schedulers.*

Hence one can decide in polynomial time whether an arbitrary ergodic scheduler exists; if yes, one can compute in polynomial time a static randomized ergodic scheduler.

Let us first concentrate on part 1. Let Θ be an ergodic scheduler. Roughly speaking, we prove that a feasible solution of the traffic LP can be constructed using (limit) frequencies of firing individual actions in $\mathcal{N}_\Theta$. Formally, given a run ω of $\mathcal{N}_\Theta$, $t \in \mathbb{R}_{\geq 0}$ and $\xi \in \bar{\Sigma}$, we denote by $O_\xi^{\leq t}(\omega)$ the number of times the action ξ is fired up to time t on ω. For memoryless Θ we have the following result.

▶ **Lemma 11.** *Assume that* Θ *is a* memoryless *ergodic scheduler. For every* $\xi \in \bar{\Sigma}$ *there is a constant* O_ξ *such that for almost all runs* ω *of* $\mathcal{N}_\Theta$ *the limit*

$$\lim_{t\to\infty} \frac{O_{\bar{\xi}}^{\leq t}(\omega)}{t}$$

exists and is equal to O_ξ*. There is* $\bar{\delta} < 1$ *such that* $\left(\bar{\delta}, O_\xi \mid \xi \in \bar{\Sigma}\right)$ *solves the traffic LP. Moreover, for every* $i \in \{1, \ldots, n\}$ *the utilization* ρ_i *of the queue* i *in* $\mathcal{N}_\Theta$ *is equal to* $\frac{\sum_{\xi\in\Sigma_i} O_\xi}{\mu_i}$.

We prove Lemma 11 in [4]. If there exists an arbitrary (i.e. possibly history-dependent) ergodic scheduler, then by Theorem 7.3.8 of [25] there exists also a memoryless (and deterministic) ergodic scheduler.[4] This fact, combined with Lemma 11, implies part 1. of Proposition 10.

Now let us concentrate on part 2. of Proposition 10.

▶ **Lemma 12.** *Any feasible solution* $\left(\bar{\delta}, \bar{\lambda}_\xi \mid \xi \in \bar{\Sigma}\right)$ *of the traffic LP with* $\bar{\delta} < 1$ induces *a static randomized ergodic scheduler whose utilization of any queue* i *is equal to* $\frac{\sum_{\xi\in\Sigma_i} \bar{\lambda}_\xi}{\mu_i}$.

Proof. We construct a static randomized scheduler Θ which chooses an action $\xi \in \Sigma_i$ for the queue i with probability

$$P_\xi = \frac{\bar{\lambda}_\xi}{\sum_{\zeta\in\Sigma_i} \bar{\lambda}_\zeta} \quad , \quad \sum_{\zeta\in\Sigma_i} \bar{\lambda}_\zeta > 0\,. \tag{5}$$

Otherwise, if $\sum_{\zeta\in\Sigma_i} \bar{\lambda}_\zeta = 0$, we may control the queue i arbitrarily because no jobs ever come to the queue. We further assume (w.l.o.g.) that such queues have been removed from the network, i.e., that P_ξ is defined using (5) for all $\xi \in \bar{\Sigma}$. Note that $\sum_{\xi\in\Sigma_i} P_\xi = 1$ for every $i \in \{1, \ldots, n\}$.

Fixing the scheduler Θ we obtain a purely stochastic branching network whose traffic equations are deficiently solvable. Formally, we define a new purely stochastic branching network $\mathcal{N}'$ with n queues with the same arrival rate, the same arrival production function and the same queue rates as $\mathcal{N}$. Further, $\mathcal{N}'$ has $R'_i = \bigcup_{\xi\in\Sigma_i} R_i(\xi)$ and the following production functions $Prob'_i$ associated to queues:

$$Prob'_i(\boldsymbol{r}) = \sum_{\xi\in\Sigma_i} P_\xi \cdot Prob_i(\xi)(\boldsymbol{r}) \quad , \quad \boldsymbol{r} \in R'_i$$

(Here we formally assume $Prob_i(\xi)(\boldsymbol{r}) = 0$ for $\boldsymbol{r} \notin R_\xi$.) The traffic equations (1) for $\mathcal{N}'$ have the following form:

$$\lambda_j \quad = \quad \alpha_j + \sum_{i=1}^{n} \lambda_i \cdot A'_{ij} \quad , \quad j \in \{1, \ldots, n\} \tag{6}$$

with

$$A'_{ij} := \sum_{\boldsymbol{r}\in R'_i} Prob'_i(\boldsymbol{r}) \cdot \boldsymbol{r}_j = \sum_{\xi\in\Sigma_i} P_\xi \sum_{\boldsymbol{r}\in R'_i} Prob_i(\xi)(\boldsymbol{r}) \cdot \boldsymbol{r}_j =$$

$$= \sum_{\xi\in\Sigma_i} \frac{\bar{\lambda}_\xi}{\sum_{\zeta\in\Sigma_i} \bar{\lambda}_\zeta} \sum_{\boldsymbol{r}\in R'_i} Prob_i(\xi)(\boldsymbol{r}) \cdot \boldsymbol{r}_j = \sum_{\xi\in\Sigma_i} \frac{\bar{\lambda}_\xi}{\sum_{\zeta\in\Sigma_i} \bar{\lambda}_\zeta} \cdot A_{\xi j}\,.$$

[4] To be formally correct, we apply Theorem 7.3.8 of [25] to the *embedded* discrete time MDP and obtain a scheduler which returns to the state $\mathbf{0}$ in finitely many steps (on average). As there are only finitely many rates in our system, this means that also the expected return time to $\mathbf{0}$ is finite.

Setting $\lambda_i := \sum_{\xi \in \Sigma_i} \bar{\lambda}_\xi$ for every $i \in \{1, \ldots, n\}$, we obtain $\lambda_i A'_{ij} = \sum_{\xi \in \Sigma_i} \bar{\lambda}_\xi A_{\xi j}$. If we put this equality into the first equation of the traffic LP, we see that $(\lambda_1, \ldots, \lambda_n)$ solves (6). Also, $\lambda_j < \mu_j$ for all $j \in \{1, \ldots, n\}$. Proposition 4 then implies that the scheduler Θ is ergodic.

Finally, let us concentrate on the utilization. Note that the utilization of any queue i is the same in $\mathcal{N}'$ as in $\mathcal{N}_\Theta$, so it suffices to concentrate on $\mathcal{N}'$. Observe that the matrix $I - A'$ is invertible by Lemma 6. This means that $(\lambda_1, \ldots, \lambda_n)$ is, in fact, the *unique* solution of (6). Then however, by Lemma 11, the utilization ρ_i of queue i in $\mathcal{N}'$ (and thus also in $\mathcal{N}_\Theta$) is equal to $\frac{\lambda_i}{\mu_i} = \frac{\sum_{\xi \in \Sigma_i} \bar{\lambda}_\xi}{\mu_i}$. ◀

To complete the proof of Proposition 10, we consider the problem of minimizing the maximal utilization $\max_i \rho_i$. Let Θ_s be a static randomized ergodic scheduler induced by a solution of the traffic LP in the sense of Lemma 12 (here we consider a solution which minimizes δ). Observe that the scheduler Θ_s minimizes $\max_i \rho_i$ among all schedulers induced by solutions of the traffic LP. However, by Lemmas 11 and 12, for every memoryless scheduler Θ there exists a static randomized scheduler induced by a solution of the traffic LP which has the same utilization of each queue as Θ. Thus Θ_s minimizes $\max_i \rho_i$ among all memoryless ergodic schedulers.

4 Conclusions

We have suggested and studied controlled branching networks, a queueing model which extends Jackson networks by nondeterministic and branching features as required to model parallel systems. Although much of the classical theory (such as product-form stationary distributions) no longer holds for controlled branching networks, we have shown that the traffic equations can be generalized. This enabled us to construct a suitable Lyapunov function which we have used to establish strong stability properties. We have shown for the controlled model that static randomized schedulers are sufficient to achieve those strong stability properties. Linear programming can be used to efficiently compute such a scheduler, which at the same time minimizes the maximal queue utilization.

Future work should include the investigation of more performance measures, e.g., the long-time average queue size. Can non-static schedulers help to minimize it?

Acknowledgements: The authors thank Javier Esparza for helpful discussions, and anonymous reviewers for valuable comments.

References

1 M. Ahmadi and S. Wong. A performance model for network processor architectures in packet processing system. In *Proceedings of the 19th IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 176–181. ACTA Press, 2007.

2 A. Azaron and S.M.T. Fatemi Ghomi. Optimal control of service rates and arrivals in Jackson networks. *European Journal of Operational Research*, 147(1):17–31, 2003.

3 G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi. *Queueing Networks and Markov Chains*. John Wiley and Sons, 2006.

4 T. Brázdil and S. Kiefer. Stabilization of branching queueing networks. Technical report, arxiv.org, 2011. Available at `http://arxiv.org/abs/1112.1041`.

5 T. Brázdil, S. Kiefer, A. Kučera, and I. Hutařová Vařeková. Runtime analysis of probabilistic programs with unbounded recursion. In *Proceedings of ICALP*, volume 6756 of *LNCS*, pages 319–331, 2011.

6 H. Chen and D. Yao. *Fundamentals of Queueing Networks*. Springer, 2001.

7 J. Daigle. *Queueing Theory with Applications to Packet Telecommunication*. Springer, 2010.

8 J.D. Deng and M.K. Purvis. Multi-core application performance optimization using a constrained tandem queueing model. *Journal of Network and Computer Applications*, In Press, Corrected Proof, 2011. DOI: 10.1016/j.jnca.2011.07.004.

9 D. Down and S.P. Meyn. Piecewise linear test functions for stability and instability of queueing networks. *Queueing Systems*, 27:205–226, 1997.

10 J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. In *LICS 2004*, pages 12–21. IEEE, 2004.

11 K. Etessami, D. Wojtczak, and M. Yannakakis. Recursive stochastic games with positive rewards. In *Proceedings of ICALP 2008*, pages 711–723, 2008.

12 K. Etessami and M. Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1):1–66, 2009.

13 K.D. Glazebrook and J. Niño-Mora. A linear programming approach to stability, optimisation and performance analysis for Markovian multiclass queueing networks. *Annals of Operations Research*, 92:1–18, 1999.

14 J.M. Harrison and R.J. Williams. Brownian models of feedforward queueing networks: Quasireversibility and product form solutions. *Annals of Applied Probability*, 2(2):263–293, 1992.

15 J.R. Jackson. Networks of waiting lines. *Operations Research*, 5(4):518–521, 1957.

16 S. Kiefer and D. Wojtczak. On probabilistic parallel programs with process creation and synchronisation. In *Proceedings of TACAS*, volume 6605 of *LNCS*, pages 296–310. Springer, 2011.

17 M.Y. Kitaev and V.V.Rykov. *Controlled queueing systems*. CRC Press, 1995.

18 P.R. Kumar and S.P. Meyn. Duality and linear programs for stability and performance analysis of queueing networks and scheduling policies. *IEEE Transactions on Automatic Control*, 42(1):4–17, 1996.

19 M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Proceedings of CAV*, volume 6806 of *LNCS*, pages 585–591, 2011.

20 N. Madan, A. Buyuktosunoglu, P. Bose, and M. Annavaram. A case for guarded power gating for multi-core processors. In *High Performance Computer Architecture (HPCA)*, pages 291–300, 2011.

21 W.A. Massey and R. Srinivasan. A heavy traffic analysis for semi-open networks. *Performance Evaluation*, 13(1):59–66, 1991.

22 S.P. Meyn and R.L. Tweedie. *Markov Chains and Stochastic Stability*. Springer, 1993.

23 M. Neuhäußer, M. Stoelinga, and J.-P. Katoen. Delayed nondeterminism in continuous-time Markov decision processes. In *Proceedings of FoSSaCS 2009*, volume 5504 of *LNCS*, pages 364–379. Springer, 2009.

24 C.H. Papadimitriou and J.N. Tsitsiklis. The complexity of optimal queueing network control. *Mathematics of Operations Research*, 24:293–305, 1994.

25 M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, 2008.

26 S. Stidham, Jr. Optimal control of admission to a queueing system. *IEEE Transactions on Automatic Control*, 30(8):705–713, 1985.

27 S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *Proceedings of TACAS*, volume 4963 of *LNCS*, pages 299–314, 2008.

28 H. Zisgen, I. Meents, B.R. Wheeler, and T. Hanschke. A queueing network based system to model capacity and cycle time for semiconductor fabrication. In *Proceedings of the 40th Conference on Winter Simulation*, pages 2067–2074, 2008.

Stronger Lower Bounds and Randomness-Hardness Trade-Offs Using Associated Algebraic Complexity Classes

Maurice Jansen and Rahul Santhanam

School of Informatics, The University of Edinburgh
Informatics Forum, 10 Crichton Street
Edinburgh, EH8 9AB, United Kingdom
`maurice.julien.jansen@gmail.com, rsanthan@inf.ed.ac.uk`

Abstract

We associate to each Boolean language complexity class $\mathcal{C}$ the algebraic class a·$\mathcal{C}$ consisting of families of polynomials $\{f_n\}$ for which the evaluation problem over $\mathbb{Z}$ is in $\mathcal{C}$. We prove the following lower bound and randomness-to-hardness results:

1. If polynomial identity testing (PIT) is in NSUBEXP then a·NEXP does not have *poly* size constant-free arithmetic circuits.
2. a·NEXP$^{\mathrm{RP}}$ does not have *poly* size constant-free arithmetic circuits.
3. For every fixed k, a·MA does not have arithmetic circuits of size n^k.

Items 1 and 2 strengthen two results due to Kabanets and Impagliazzo [7]. The third item improves a lower bound due to Santhanam [11].

We consider the special case low-PIT of identity testing for (constant-free) arithmetic circuits with low formal degree, and give improved hardness-to-randomness trade-offs that apply to this case.

Combining our results for both directions of the hardness-randomness connection, we demonstrate a case where derandomization of PIT and proving lower bounds are *equivalent*. Namely, we show that low-PIT $\in$ i.o-NTIME$[2^{n^{o(1)}}]/n^{o(1)}$ if and only if there exists a family of multilinear polynomials in a·NE/*lin* that requires constant-free arithmetic circuits of super-polynomial size and formal degree.

1998 ACM Subject Classification F.1.3 Complexity Measures and Classes.

Keywords and phrases Computational Complexity, Circuit Lower Bounds, Polynomial Identity Testing, Derandomization.

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.519

1 Introduction

In this paper we study the arithmetic circuit complexity of families of multivariate polynomials $\{f_n\}$ in terms of the computational hardness of the underlying evaluation problem. Towards this end we associate to each Boolean language complexity class $\mathcal{C}$ the class a·$\mathcal{C}$ consisting of all families of polynomials $\{f_n\}$ with integer coefficients, such that given an integer input tuple x to f_n, an integer i and a bit b, it can be decided within the resources of the class $\mathcal{C}$ whether the ith bit of $f_n(x)$ equals b. We restrict the number of variables, the degree, and the bit size of coefficients of such families to be polynomially bounded in n (See Section 2 for the formal definition). We note that a similar notion was suggested by Koiran and Perifel [9].

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 519–530

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


One of our main motivations is to find an elegant way to state "hybrid" results involving Boolean and arithmetic circuit lower bounds, such as the trade-offs of Kabanets and Impagliazzo [7] or the lower bound of Santhanam [11]. These are examples where people, perhaps unknowingly, have been proving lower bounds and randomness-hardness tradeoffs geared towards associated algebraic classes, while in our opinion lacking the proper language to describe, and consequently interpret, the results. The a·$\mathcal{C}$ notions provides a language for succinctly expressing these results, and leads to natural questions for making improvements. Consequently we have strengthened several important results from the literature.

A prime example of the above situation is the celebrated theorem by Kabanets and Impagliazzo [7], which says that if polynomial identity testing (PIT) is in NSUBEXP, then either NEXP $\not\subseteq$ P/*poly*, or permanent does not have poly-size arithmetic circuits. PIT is the problem of deciding for a given arithmetic circuit Φ whether it computes the zero polynomial. We refer to a recent survey by Saxena [12] for more on this problem. The quoted theorem tells us that derandomization of polynomial identity testing yields lower bounds of *some* sort. However, it doesn't tell us whether these will be Boolean lower bounds or arithmetic lower bounds. We make the observation[1] that the theorem by Kabanets and Impagliazzo is equivalent to the statement PIT $\in$ NSUBEXP $\Rightarrow$ a·NEXP/*lin* $\not\subseteq$ ASIZE$'$(*poly*). Here ASIZE$'$(*poly*) denotes the class of polynomial families $\{f_n\}$ computable by constant-free arithmetic circuit of size *poly*(n) using addition, multiplication, and division by computed constants (See Section 2). Hence, to answer the above question, putting PIT in NSUBEXP gives *arithmetic* lower bounds for families of polynomials that can be evaluated in NEXP with linear advice. A natural improvement to the result would be to drop the linear advice. We show that this can indeed be done[2], resulting in the following stronger theorem:

▶ **Theorem 1.** PIT $\in$ NSUBEXP $\Rightarrow$ a·NEXP $\not\subseteq$ ASIZE$'$(*poly*).

In the above, on the right hand side, the associated algebraic class gives us a measure of the explicitness of the lower bound. We have improved this explicitness from evaluable in NEXP/*lin* down to evaluable in NEXP. As it is generally undesirable to have non-uniform dependencies appearing in the explicitness measure of a lower bound, the main significance of our result is that we have managed to remove the non-uniformity.

Similar to Theorem 1 we observe that Theorem 5.2 of Ref. [7], which states that either NEXP$^{\mathrm{RP}}$ $\not\subset$ P/*poly* or Permanent does not have poly-size arithmetic circuits, is equivalent to the statement that a·NEXP$^{\mathrm{RP}}$/*lin* $\not\subseteq$ ASIZE$'$(*poly*). We also improve the explicitness of this lower bound and obtain that

▶ **Theorem 2.** a·NEXP$^{\mathrm{RP}}$ $\not\subseteq$ ASIZE$'$(*poly*).

Furthermore, we improve a theorem by Santhanam [11] which states that for every k, either MA $\not\subseteq$ SIZE(n^k), or there exists a family polynomial $\{f_n\}$, whose graph is decidable in MA, that is not in ASIZE(n^k). We show the following stronger result:

▶ **Theorem 3.** *For every k,* a·MA $\not\subseteq$ ASIZE(n^k).

The above results demonstrate the usefulness of the a·$\mathcal{C}$ notion. There are further reasons why the notion is worth exploring. It gives a way of bringing uniformity into the algebraic complexity setting. Note that traditional algebraic complexity classes such as VP and VNP

[1] A proof will appear in the full version of this paper.

[2] An obvious way to do this would be to 'just' show that a·NEXP $\subseteq$ ASIZE$'$(*poly*) $\Rightarrow$ a·NEXP/*lin* $\subseteq$ ASIZE$'$(*poly*). It is not clear whether this is true.

are inherently non-uniform. We also feel the notion could facilitate more interactions of techniques from structural complexity and algebraic complexity. Given how few lower bound techniques we have available, and given the well-known barriers such as natural proofs and algebrization to finding new ones, we need to make the best use of the ones we have. The recent lower bounds success of Williams [17] is an instructive example of how known techniques from different domains can be combined to give an interesting new result.

In general, one might ask for any known separation $\mathcal{C} \not\subseteq \mathcal{D}$ in the Boolean world whether it can be strengthened to show that a·$\mathcal{C} \not\subseteq$ a·$\mathcal{D}$. Note that this would indeed be a strengthening as $\mathcal{C} \subseteq \mathcal{D}$ trivially implies a·$\mathcal{C} \subseteq$ a·$\mathcal{D}$. Arithmetic analogues of time hierarchy results, eg., a·DTIME$[n^2] \subsetneq$ a·DTIME$[n^3]$ and a·NTIME$[n^2] \subsetneq$ a·DTIME$[n^3]$ can be proved quite easily using the fact that the separation can be witnessed by a unary language. However, we don't know whether arithmetic analogues of results such as Williams' lower bound hold. There could be a connection between proving the arithmetic analogue of a Boolean result and whether the techniques used to prove the Boolean result algebrize in the sense of Aaronson and Wigderson [1]. We have not properly explored this yet.

One of the advantages of using associated algebraic classes is that this enables us to derive tighter hardness-randomness trade-offs. This is especially striking for the case of the low-formal-degree polynomial identity testing problem (low-PIT). We define low-PIT as the special case of PIT for circuits Φ whose formal degree $deg(\Phi)$ is less than or equal to the size $|\Phi|$. Formal degree is a syntactic notion, easily computed for a circuit (See Section 2). Examples of types of circuits that automatically satisfy the degree restriction $deg(\Phi) \leq |\Phi|$ are formulas and skew-circuits, the latter being equivalent to algebraic branching programs. This makes low-PIT an important special case of the general problem. We show that we can specialize Theorem 1 to obtain the following low-degree version:

▶ **Theorem 4.** *low*-PIT $\in$ NSUBEXP $\Rightarrow$ a·NEXP $\not\subseteq$ ASIZEDEG$'(poly)$.

In the above ASIZEDEG$'(poly)$ is the class of families of polynomials $\{f_n\}$ computable by constant-free arithmetic circuits of size $poly(n)$ and formal degree $poly(n)$ (The 'prime' indicates that we allow a single division by a previously computed constant at the output gate).

For the special case of low-PIT, we also make progress on trade-offs that go in the opposite direction. Namely, we show that derandomization can be achieved under weaker hardness assumptions than was known previously. For example using our techniques we can prove the following theorem:

▶ **Theorem 5** (Proof to appear in full version)**.** *Suppose there exists a family* $\{p_n\} \in$ ml·NEXP *with* $\{p_n\} \notin$ i.o-ASIZEDEG$'(n^{e(n)})$*, where* $e(n)$ *is a monotone non-decreasing time constructible function with* $e(n) = \omega(1)$*. Then low*-PIT $\in$ NTIME$[2^{n^{o(1)}}]$.

In the above, ml·NEXP is the subclass of a·NEXP consisting of all families $\{f_n\}$, where each f_n is multilinear. The key improvement[3] that we make here over the techniques of Ref. [7], is that we can work with ASIZEDEG$'$-hardness instead of ASIZE$'$-hardness in case we only need to cater for low-PIT. To achieve such improved trade-offs, we prove a so-called root extraction lemma (Lemma 19) that is formal-degree efficient. This lemma, which is of independent interest, is subsequently combined with the framework of Ref. [7]. As an additional twist, we start with a hardness assumption in terms of an associated algebraic class.

[3] See some remarks about the difference in Section 2.

Finally, combining our results for both directions of the hardness vs. randomness connection, the work of this paper culminates with the following theorem, which demonstrates a setting where derandomization of PIT and proving lower bounds are *formally equivalent*:

▶ **Theorem 6.** *There exists a family* $\{p_n\} \in \text{ml·NE}/lin$ *with* $\{p_n\} \notin \text{ASIZEDEG}'(s(n))$, *for* $s(n) = n^{\omega(1)}$ *if and only if low-*PIT $\in \text{i.o-NTIME}[2^{r(n)}]/r(n)$, *for* $r(n) = n^{o(1)}$.

In the past there have been several authors claiming partial converses to randomness-hardness theorems involving PIT. Our paper is the first where an actual equivalence is being observed. In this the associated algebraic classes play a central role, which we offer as further evidence of the importance of this notion.

2 Preliminaries

Let $\text{NSUBEXP} = \cap_\epsilon \text{NTIME}[2^{n^\epsilon}]$. We define $\text{SIZE}(s(n))$ to be the class of all languages in $\{0,1\}^*$ computable by Boolean circuits of size $s(n)$. A (division-free) arithmetic circuit over some field $\mathbb{F}$ and a set of variables $X = \{x_1, x_2, \ldots, x_n\}$ is given by a labeled directed acyclic graph. Nodes of in-degree zero are labeled with elements of $X \cup \mathbb{F}$. Other nodes are labeled by $+$ or $\times$. To each node, a.k.a. gate, we can associate a polynomial $\in \mathbb{F}[X]$, defined inductively in the obvious way. If constant-labels are restricted to be in $\{-1, 0, 1\}$ the circuit is called constant-free. For the size of an arithmetic circuit we count the number of wires. We define $\text{ASIZE}(s(n))$ to be the class of all families of polynomials $\{f_n\}$ with integer coefficients that have constant-free arithmetic circuits of size $s(n)$. We let $\text{ASIZE}'(s(n))$ be the class obtained from $\text{ASIZE}(s(n)$ by allowing one single division at the output gate by an integer $a \neq 0$, where a has been computed by the circuit. We remark that due to a result by Strassen [14] on avoidance of divisions, cf. Theorem 2.17 and Corollary 3.9 in [7], a family of polynomials $\{f_n\}$ of $poly(n)$ degree can be computed by an arithmetic circuit of $poly(n)$ size *with arbitrary use of division gates* iff $\{f_n\} \in \text{ASIZE}'(poly(n))$. We define "infinitely often" versions of these classes in the obvious way. For example $\text{i.o-ASIZE}(s(n))$ is the class of families $\{f_n\}$ such that for infinitely many n, f_n can be computed by a size $s(n)$ circuit.

$\text{ASIZEDEG}(s(n))$ is obtained from $\text{ASIZE}(s(n))$ by adding the restriction that formal degree of the circuit is bounded by $s(n)$ as well. Formal degree is defined inductively as follows. For input gates, regardless of their label, formal degree is 1. Formal degree of an addition gate is taken to be the maximum of the formal degree of its inputs. For multiplication gates one takes the sum of formal degrees of its inputs. We define the class $\text{ASIZEDEG}'(poly)$ to be the class of families of polynomials $\{f_n\}$ with integer coefficients such that there exist families $\{g_n\}$ and $\{c_n \in \mathbb{Z}\}$ in $\text{ASIZEDEG}(poly)$ with $f_n = g_n/c_n$, for each n.

Families in $\text{ASIZE}(poly)$ can have super-polynomial degree, e.g. x^{2^n} can be computed with $n-1$ repeated multiplications. We like to point out that in general for a family $\{f_n\} \in \text{ASIZE}(poly)$ with $deg(f_n) = n^{O(1)}$ *it is not known* whether $\{f_n\} \in \text{ASIZEDEG}(poly)$. In particular it is a fallacy to think the well-known trick of computing degree components separately at every gate in the circuit proves this, as was pointed[4] out by Bürgisser [3], cf. [8]. Namely, this construction requires a model where arbitrary constants can be used by the circuit at unit cost. A similar remark can be made for the classes $\text{ASIZEDEG}'(poly)$

[4]The class $\text{ASIZEDEG}(poly)$ is known as VP^0 in the literature, whereas $\text{ASIZE}(poly)$ corresponds to families of polynomials with τ-complexity $poly(n)$, cf. Ref. [9].

and ASIZE′(*poly*), in which case it is perhaps more obvious that computing components separately does not help, since one of the given circuits only computes a constant.

We define the language corresponding to the polynomial identity testing problem PIT = $\{\Phi : \Phi$ is a division-free constant-free arithmetic circuit such that $\Phi \equiv 0\}$. Similarly, we define low-PIT to be the following language

$\{\Phi : \Phi$ is a division-free constant-free arithmetic circuit of formal degree $\leq |\Phi|$ and $\Phi \equiv 0\}$.

We make use of the well-known Schwartz-Zippel-deMillo-Lipton Lemma.

▶ **Lemma 7** ([4, 13, 18]). *Let A be an arbitrary nonempty subset of the field $\mathbb{F}$. Then for any nonzero polynomial $f \in \mathbb{F}[X]$ of degree d, $\Pr[f(a_1, a_2, \ldots, a_n) = 0] \leq \frac{d}{|A|}$, where the a_i's are picked independently and uniformly at random from A.*

We use several easily proved propositions.

▶ Proposition 1. A constant-free division-free arithmetic circuit of size s and formal degree d without variables computes an integer constant of absolute value at most 2^{ds}.

By hard-wiring inputs we obtain the following corollary:

▶ **Corollary 8.** *For a constant-free division-free arithmetic circuit of size s and formal degree d computing a polynomial $f(x_1, x_2, \ldots, x_n)$, if we evaluate f on integers $a_1, \ldots, a_n$ of at most B bits, then $|f(a_1, a_2, \ldots, a_n)| \leq 2^{O(dsB^2)}$.*

▶ Proposition 2 (Multilinear Extension over $\mathbb{Z}$). Let $f : \{0,1\}^n \to \{0,1\}$ be a Boolean function. Define $F(x_1, \ldots, x_n) = \sum_{a \in \{0,1\}^n} f(a) \prod_{i \in [n]} (1 - x_i + a_i(2x_i - 1))$. Then F is a multilinear polynomial with integer coefficients that coincides with f on $\{0,1\}^n$. Furthermore, F is the unique polynomial with these properties.

We now get to the central definition of this paper.

▶ **Definition 9.** Let $\mathcal{C}$ be a language complexity class. Corresponding to $\mathcal{C}$ we have the associated algebraic class a·$\mathcal{C}$ which is given by the collection of all polynomial families $\{f_n\}$ defined in $m(n) = n^{O(1)}$ variables of degree $poly(n)$ having integer coefficients of $poly(n)$ bit size such that the evaluation language

$$E(\{f_n\}) := \{(1^n, a_1, a_2, \ldots, a_{m(n)}, i, b) : \text{the } i\text{th bit of } f_n(a_1, a_2, \ldots, a_{m(n)}) \text{ equals } b\} \in \mathcal{C},$$

where i, $a_1, a_2, \ldots, a_{m(n)} \in \mathbb{Z}$ are given in binary. We denote the subclass of a·$\mathcal{C}$ consisting of families of multilinear polynomials by ml·$\mathcal{C}$.

Typically for a complexity class $\mathcal{C}$ we will have the complementation property that a·$\mathcal{C}$ = a·($\mathcal{C} \cap$ co$\mathcal{C}$). This is due to the inclusion of the bit b in the definition of the evaluation language. We have the following property in particular:

▶ Proposition 3. a·NEXP = a·(NEXP ∩ coNEXP).

Namely, given a NEXP-machine M deciding $E(\{f_n\})$ for some family of polynomials $\{f_n\}$, one can simulate M on inputs $(1^n, a_1, a_2, \ldots, a_{m(n)}, i, b)$ and $(1^n, a_1, a_2, \ldots, a_{m(n)}, i, 1-b)$. In these nondeterministic simulations one finds at most one of them accepting, and it is guaranteed there exists at least one such path. For all paths where an accept is found, the machine knows exactly whether $(1^n, a_1, a_2, \ldots, a_{m(n)}, i, b) \in E(\{f_n\})$. This means that we have a nondeterministic exponential time *flag machine* for computing the characteristic function of $E(\{f_n\})$, which implies that $E(\{f_n\}) \in$ NEXP ∩ coNEXP. Recall that a flag machine sets a flag bit and produces output. If the flag is 0 this means on the given path no output is produced. If the flag is 1 it signals the output is valid. To compute a function, all flag = 1 paths must produce the same value, and there must be at least one such path.

3 Improved Lower Bounds from Derandomization of PIT

In this section, in order to avoid ambiguity we use a new variable N to indicate the input length for Boolean complexity classes. For example, $\Sigma_4\text{TIME}[N]$ is the class of all languages decidable by Σ_4-machines in time $O(N)$ for inputs of size N. We will prove Theorem 2, and the following theorem (which implies Theorem 1):

▶ **Theorem 10.** PIT $\in$ NSUBEXP $\Rightarrow$ ml·NEXP $\not\subseteq$ ASIZE$'(poly)$.

We first establish fixed polynomial size arithmetic circuit lower bounds for a·PH.

▶ **Theorem 11.** *For any fixed k, there exists $\{f_n\} \in$ a·PH with $\{f_n\} \notin$ i.o-ASIZE$'(n^k)$. Furthermore, each f_n is multilinear in n variables, has degree $3k$ and has coefficients in $\{0,1\}$.*

Proof. For simplicity we show a fixed size lower bound in terms of ASIZE instead of ASIZE$'$. The proof is easily modified to yield the more general statement. There are $2^{O(n^{2k})}$ arithmetic circuits of size at most n^k. Consider the class $\mathcal{F}$ of homogeneous multilinear polynomials in n variables of degree $3k$ with $0,1$ coefficients. Then $|\mathcal{F}| = 2^{\binom{n}{3k}}$. Hence there exists $f_n \in \mathcal{F}$ that is not in ASIZE(n^k). Our goal is to find it 'in PH'.

Let $\mathcal{C}$ be the class of arithmetic circuits corresponding to $\mathcal{F}$, where we just represent in the $\Sigma\Pi$-form, i.e. a sum of monomials. We can fix some representation of $\mathcal{C}$ by strings of length $O(n^{3k})$. Our goals is to find the lexicographically least circuit $\Phi \in \mathcal{C}$ such that *for all* arithmetic circuits Ψ of size n^k, $\Phi - \Psi \not\equiv 0$. Define the language L to consist of tuples $(1^n, < \Phi >)$ with the property that for all circuits Ψ of size n^k, $\Phi - \Psi \not\equiv 0$, where $< \Phi >$ to denotes the string encoding of Φ. Checking $\Phi - \Psi \not\equiv 0$ is a coRP predicate. This implies that L is in coNP^{RP}. On input 1^n, using binary search and making *existential* queries to L, one can find the lexicographically least Φ of size $O(n^{3k})$ such that $(1^n, < \Phi >) \in L$ in $\text{FP}^{\text{NP}^{\text{coRP}^{\text{RP}}}}$. Define f_n to be the polynomial computed by this Φ. Once the sum of monomials representations of f_n is known, evaluations is *poly*-time computable for integer inputs. Hence we obtain that $E(\{f_n\}) \in$ PH. ◀

From the proof of Theorem 11 we can conclude that the following lemma is true:

▶ **Lemma 12.** *There exists a constant $c_1 \in \mathbb{N}$, such that for any $k \geq 1$, there exists $\{f_n\} \in$ ml· $\Sigma_4\text{TIME}[N^{c_1 k}]$ with $\{f_n\} \notin$ i.o-ASIZE$'(n^k)$.*

Namely, to describe an algorithm for $E(\{f_n\})$, consider an input $(1^n, a_1, \ldots, a_n, i, b)$ of size N. The proof of Theorem 11 shows that we can first find in $\Sigma_4\text{TIME}[poly(n^{3k})]$ a sum-of-monomials description of a polynomial f_n of size $O(n^{3k})$ that requires size n^k. After that we evaluate $f(a_1, \ldots, a_n)$, which can be done in time $poly(N^{3k})$ given this simple representation of f_n. We get that the total overhead for deciding $E(\{f_n\})$ is $\Sigma_4\text{TIME}[poly(N^k)]$. One now easily derives the following lemma:

▶ **Lemma 13.** *There exists a constant $c_2 \in \mathbb{N}$, such that for any $k \geq 1$, there exists $\{f_n\} \in$ ml·$\text{DTIME}^{0,1\text{-}Perm^{[1]}}[N^{c_2 k}]$ with $\{f_n\} \notin$ i.o-ASIZE$'(n^k)$.*

Proof. By Toda's theorem [15] and Valiant's Completeness result [16], we know that there exists an absolute constant $b \in \mathbb{N}$ so that for every $k \in \mathbb{N}$, $\Sigma_4\text{TIME}[N^k] \subseteq \text{DTIME}^{0,1\text{-Perm}^{[1]}}[N^{bk}]$. Let c_1 be the constant given by Lemma 12. We get that $\Sigma_4\text{TIME}[N^{c_1 k}] \subseteq \text{DTIME}^{0,1\text{-Perm}^{[1]}}[N^{bc_1 k}]$. Hence the lemma holds for $c_2 = bc_1$. ◀

We use the following lemma by Kinne, van Melkebeek and Shaltiel:

▶ **Lemma 14** (Claim 5 in [5]). *There exists a constant d such that the following holds for any functions $a(\cdot)$ and $t(\cdot)$ with $a(\cdot)$ time-constructible and $t(\cdot)$ monotone. If* PIT $\in$ NTIME$(t(N))$ *and* $\{per_n\} \in$ ASIZE$'(a(n))$, *then* DTIME$^{0,1\text{-}Perm[1]}[N] \subseteq$ NTIME$[t(N \cdot \log^d N \cdot a(\sqrt{N}))]$.

We are now ready to prove Theorem 10.

Proof. *(Theorem 10)* We are done if $\{per_n\} \notin$ ASIZE$'(poly)$, so assume that ASIZE$'(\{per_n\}) \leq n^\ell$, for $\ell \in \mathbb{N}$. Consider arbitrary $k \geq 1$. Combining Lemmas 13 and 14, we obtain that for any monotone function $t(\cdot)$, if PIT $\in$ NTIME$[t(N)]$, then ml·NTIME$[t(N^{c_2 k} \cdot \log^d N^{c_2 k} \cdot N^{c_2 \ell k/2})] \not\subseteq$ ASIZE$'(n^k)$. As we are assuming that PIT $\in$ NSUBEXP, if we apply this with $t(N) = 2^{N^\epsilon}$, for small enough ϵ, we get that ml·NTIME$[2^N] \not\subseteq$ ASIZE$'(n^k)$. Since k was arbitrary, we get that ml·NTIME$[2^N] \not\subseteq$ ASIZE$'(poly)$, which implies that ml·NEXP $\not\subseteq$ ASIZE$'(poly)$. ◀

Next we move on to the proof of Theorem 2.

Proof. *(Theorem 2).* Suppose that a·NEXP$^{\text{RP}} \subseteq$ ASIZE$'(poly)$. Then a·EXP $\subseteq$ ASIZE$'(poly)$. We claim that this implies that EXP $\subseteq$ SIZE$(poly)$. Let $L \in$ EXP be any language. We will show that $L \in$ SIZE$(poly)$. Since we can evaluate multilinear extensions (Proposition 2) of characteristic functions of EXP languages within EXP itself, we get $\{F_n\}$ in a·EXP, where F_n is the multilinear extension of χ_L on $\{0,1\}^n$. We get that $\{F_n\} \in$ ASIZE$'(poly)$. This means that we have constant-free (division-free) arithmetic circuits Φ_1 and Φ_2 of size at most $p(n) = n^{O(1)}$, such that Φ_2 does not contain variables and computes some nonzero constant $c \in \mathbb{Z}$. Furthermore, if Φ_1 computes G_n then it holds that $G_n = c \cdot F_n$. For input $a \in \{0,1\}^n$, $F_n(a) \in \{0,1\}$, which means for such inputs $G_n(a) \in \{0, c\}$. We want to evaluate Φ_1 modulo some prime number q that does not divide c. This will tell us $\chi_L(a)$. We have that $|c| \leq 2^{2^{p(n)}}$ due to Proposition 1. This means that c has at most $2^{p(n)}$ prime factors. Hence, using the Prime Number Theorem there exists a prime number q of $p(n)^2$ bits, provided n is large enough, that does not divide c. As our task is to show only the non-uniform upper bound $L \in$ SIZE$(poly)$, mere existence of this number q suffices for our purposes, as we can hardcode it into the Boolean circuit simulating Φ_1 and Φ_2. Hence EXP $\subseteq$ SIZE$(poly)$.

Babai, Fortnow, Lund [2] prove that EXP $\subseteq$ SIZE$(poly) \Rightarrow$ EXP = MA. So we get that EXP = MA. Also, because easily $\{per_n\} \in$ a·NEXP$^{\text{RP}}$, we have that $\{per_n\} \in$ ASIZE$'(poly)$. This implies that P$^{\#\text{P}} \subseteq$ NP$^{\text{RP}}$, cf. Lemma 5.3 in Ref. [7]. By Toda's Theorem [15], MA $\subseteq$ P$^{\#\text{P}}$. Hence we obtain that EXP = MA $\subseteq$ NP$^{\text{RP}}$. By padding this implies that EEXP $\subseteq$ NEXP$^{\text{RP}}$. Hence a·EEXP $\subseteq$ a·NEXP$^{\text{RP}} \subseteq$ ASIZE$'(poly)$. This is a contradiction. One can easily deduce that a·EEXP $\not\subseteq$ i.o-ASIZE$'(n^{\log n})$ by observing that Lemma 12 also holds if we allow k to depend on n as $k(n) = \lceil \log n \rceil$. ◀

We can specialize Lemma 14 so that we replace the condition "PIT $\in$ NTIME$(t(N))$ and $\{per_n\} \in$ ASIZE$'(a(n))$" by "low-PIT $\in$ NTIME$(t(N))$ and $\{per_n\} \in$ ASIZEDEG$'(a(n))$". This yields the following theorem (which implies Theorem 4):

▶ **Theorem 15.** *low*-PIT $\in$ NSUBEXP $\Rightarrow$ ml·NEXP $\not\subseteq$ ASIZEDEG$'(poly)$.

4 Stronger Fixed Size Lower Bounds for MA

As the result we aim to strengthen puts somewhat different constraints on constants appearing in arithmetic circuits compared to what we have seen so far, we make the following provisional definition. Let $\text{ASIZE}^{\text{free}}(s(n))$ denote the class obtained from $\text{ASIZE}(s(n))$ by granting the underlying circuits arbitrary constant labels $\in \mathbb{Z}$. Similar to Theorem 11 we have the following theorem.

▶ **Theorem 16** (Proof will appear in full version). *$\forall k \exists \{f_n\} \in$ a·PH with $\{f_n\} \notin$ i.o-$\text{ASIZE}^{free}(n^k)$.*

We want to strengthen Theorem 1.4 of [11], which we can reformulate it in our terminology as follows:

▶ **Theorem 17** ([11]). *For every k, either 1) $\text{MA} \not\subseteq \text{SIZE}(n^k)$, or 2) $\text{a·MA} \not\subseteq \text{ASIZE}^{free}(n^k)$.*

We will show that for every k, the second item holds by itself. Let us briefly remark on a technical issue related to this reformulation. For $\{f_n\}$, where f_n is a integer polynomial over n variables, Ref. [11] uses the notion $Gh(\{f_n\}) = \{(\vec{x}, v) | f_n(\vec{x}) = v\}$, and proves that for every k, either $\text{MA} \notin \text{SIZE}(n^k)$, or there exists $\{f_n\} \notin \text{ASIZE}^{\text{free}}(n^k)$ with $Gh(\{f_n\}) \in \text{MA}$. We prefer to work with the evaluation language $E(\{f_n\})$ instead of $Gh(\{f_n\})$. One can observe that the argument we give to strengthen Theorem 17 can be easily modified to work with $Gh(\cdot)$ instead. Consider the following proposition:

▶ Proposition 4. If $\{per_n\} \in \text{ASIZE}^{\text{free}}(poly)$, then 1) 0, 1-permanent of an $n \times n$ matrix over $\mathbb{Z}$ can be computed with $poly(n)$ size Boolean circuits, and 2) $\text{PH} \subseteq \text{MA}$.

For the above, it is argued in Ref. [11], proof of Theorem 1.4, that the first item follows from $\{per_n\} \in \text{ASIZE}^{\text{free}}(poly)$, and that the second item follows from the first. The following theorem implies Theorem 3 from the introduction:

▶ **Theorem 18.** *For any fixed k, there exists $\{f_n\} \in \text{a·MA}/\text{ASIZE}^{free}(n^k)$.*

Proof. We show that Item 2 of Theorem 17 holds by itself. For this, we indicate how the proof of Theorem 1.4 in Ref. [11] must be modified. This proof conditions on the predicate $\{\text{per}_n\} \in \text{ASIZE}^{\text{free}}(poly)$. If this is not true, the proof there can easily be modified to use $E(\cdot)$ instead of $Gh(\cdot)$, which then yields the statement of the theorem. Otherwise, suppose that $\{\text{per}_n\} \in \text{ASIZE}^{\text{free}}(poly)$. By Proposition 4 we have that $\text{PH} \subseteq \text{MA}$. The latter implies that $\text{a·PH} \subseteq \text{a·MA}$. Hence in this case Item 2 holds also, due to Theorem 16. ◀

5 A Characterization of Derandomization for low-PIT

We will use the algebraic hardness-to-randomness framework of Ref. [7]. The refinement that we make here is to show that it suffices to start with a weaker[5] $\text{ASIZEDEG}'$-hardness assumption rather than ASIZE'-hardness, in case we only need to cater for low-PIT.

For a polynomial $f(x, y) \in \mathbb{F}[x_1, \ldots, x_n, y]$ and $p(x) \in \mathbb{F}[x_1, \ldots x_n]$, $f_{|y=p}$ denotes the polynomial obtained by substituting p for y in f. We will also write this polynomial as $f(x, p)$. In case $f_{|y=p} = 0$, we say that p is a *root* of f for y. The following is our degree-efficient root extraction lemma:

[5] See Section 2 for some remarks pertaining to these measures when dealing with families of *poly*-degree.

▶ **Lemma 19.** *Suppose that $f \in \mathbb{Z}[x_1, \ldots, x_n, y]$ is a nonzero polynomial computed by a division-free constant free arithmetic circuit of size s and formal degree D. Suppose that $p \in \mathbb{Z}[x_1, \ldots, x_n]$ is a root of f for y. Then there exist constant-free division-free arithmetic circuits Φ_1 and Φ_2 of size and formal degree bounded by $poly(n, s, D, L)$ such that the following are true:*

1. *Φ_1 computes a polynomial $q \in \mathbb{Z}[x_1, \ldots, x_n]$.*
2. *Φ_2 does not contain variables. It computes a nonzero constant $c \in \mathbb{Z}$.*
3. *It holds that $c \cdot p = q$.*
4. *L bounds the maximum bit size of $p(x)$ on $\{0, 1, \ldots, d_p d_f\}^n$, where d_f and d_p are the degrees of f and p, respectively.*

The proof of the above lemma follows by analyzing the degree blow-up in the root extraction method of Ref. [6]. As this procedure involves Newton iteration it is a priori not at all clear that formal-degrees are well-behaved, but this turns out to be true. The proof will appear in the full version of this paper. We continue towards our hardness-to-randomness trade-offs. First we need the following lemma:

▶ **Lemma 20** (Nisan-Wigderson Design [10])**.** *Let n, m be integers with $n < 2^m$. There exists a family of sets $S_1, S_2, \ldots, S_n \subseteq [\ell]$, such that 1) $\ell = O(m^2/\log n)$, 2) For each i, $|S_i| = m$, and 3) For every $i \neq j$, $|S_i \cap S_j| \leq \log n$. Furthermore, the above family of sets can be computed deterministically in time $poly(n, 2^\ell)$.*

Define NW^p as follows. For parameters ℓ, m, n, construct the set system $S_1, S_2, \ldots, S_n$ as in Lemma 20. Then for $a_1, a_2, \ldots, a_\ell \in \mathbb{F}$, and a polynomial p in m variables, $NW^p(a) = (p(a_{|S_1}), p(a_{|S_2}), \ldots, p(a_{|S_n}))$. The following lemma is derived from Lemma 19 using a hybrid argument, cf. Lemma 7.6 in [7]:

▶ **Lemma 21.** *Let n and m be integers with $n < 2^m$ and $m < n$. Suppose we are given a nonzero polynomial $f \in \mathbb{Z}[y_1, \ldots, y_n]$ of degree d_f and a multilinear polynomial $p \in \mathbb{Z}[x_1, \ldots, x_m]$ with coefficients of bit size at most m^e, for some integer constant $e \geq 1$. Assume that f can be computed by a division free constant-free arithmetic circuit of size s and formal degree D. Let $S \subseteq \mathbb{Z}$ be any set of size $|S| > d_f m$, and let ℓ be given by Lemma 20. Suppose that $f(NW^p(a)) = 0$ for all $a \in S^\ell$. Then there exists $q \in \mathbb{Z}[x_1, \ldots, x_m]$ and $c \in \mathbb{Z}/\{0\}$ such that $p = q/c$, where q and c can be computed by constant-free division-free arithmetic circuits of size and formal degree $poly(n, m^e, s, D)$.*

A proof of the above lemma will be included in the full version of the paper. Our first trade-off is as follows:

▶ **Theorem 22.** ml·NEXP $\not\subseteq$ ASIZEDEG$'(poly(n)) \Rightarrow$ *low*-PIT $\in \bigcap_{\epsilon>0}$ i.o-NTIME$[2^{N^\epsilon}]$.

Proof. Consider a family $\{p_m\} \in$ ml·NEXP that is not in ASIZEDEG$'(poly)$. By reindexing we can assume wlog. that p_m is defined over m variables. Let e be such that coefficients of p_m are at most m^e bits. We have that for every k, there exist infinitely many m such that p_m cannot be written as $p_m = f_m/c_m$, where f_m and $c_m \in \mathbb{Z}/\{0\}$ are computed by constant-free division-free arithmetic circuits of size and formal degree at most m^k. The $m \in \mathbb{Z}$ that satisfy this property we call the *good indexes for* k. We use the fact that a·NEXP = a·(NEXP $\cap$ coNEXP). This means that we there exists a constant d and a nondeterministic flag machine M running in time $2^{(n')^d}$ for inputs of size n' that can compute the characteristic function of $E(\{p_m\})$ on a given input, cf. Proposition 3.

Let c_0 be an absolute constant that bounds the overhead of Lemma 21, in the sense that for the case $n = s = D$ we can write an upper bound of $n^{c_0} m^{ec_0}$ for the bound

$poly(n, m^e, s, D)$ given by the lemma. We will describe an i.o-NSUBEXP algorithm for low-PIT. Let Φ be a constant-free (division-free) arithmetic circuit of size N computing f. First we check that the formal degree of Φ is bounded by N, if not reject.

Let $m = \lfloor N^{1/r} \rfloor$, where r is chosen arbitrarily large. We claim that for infinitely many input lengths N the following test property holds: for every constant-free arithmetic circuit Ψ of size N, $\Psi \equiv 0 \Leftrightarrow (\forall a \in S^\ell), \Psi(NW^{p_m}(a)) = 0$, where $S = [Nm+1]$ with $\ell = O(m^2/\log N)$ taken according to Lemma 20. This follows from Lemma 21. Namely, let $k = c_0(r+e)$ and let $\mathcal{M}$ be the set of good indexes for k. Then $\mathcal{M}$ is an infinite set. Consider input lengths N of the form $N = (N')^r$, where $N' \in \mathcal{M}$. For such N, we set $m = N'$. The test property can only be violated if for some Ψ of size N we have that $\Psi \not\equiv 0$, while $(\forall a \in S^\ell), \Psi(NW^{p_m}(a)) = 0$. By Lemma 21 we obtain that p_m can be written as $p_m = f_m/c_m$, for f_m and $c_m \in \mathbb{Z}/\{0\}$ that are computed by constant-free arithmetic circuits of size and formal degree at most $N^{c_0} m^{c_0 e} = (m)^{c_0(r+e)} = m^k$. We know the latter does not hold for $m \in \mathcal{M}$.

We continue the description of the algorithm. We produce a set H to sample Φ with, namely we take H to be the output of $NW^{p_m}(\cdot)$ on S^ℓ. We have that $|H| = (Nm+1)^\ell = 2^{O(N^{2/r}\log N)}$. We do simulations of the machine M for $E(\{p_m\})$ to get all the bits of all the evaluations of $p_m(\cdot)$ on S^ℓ. As p_m is multilinear with coefficients of bit length at most m^e, we can bound the bit size of any such evaluation by $O(m^e \log(Nm))$. This means that the inputs $(1^m, a_1, \ldots, a_m, i, b)$ that we simulate M on, have bit size $O(m \log(Nm))$ (recall that i is given in binary). The simulation for a single such input thus costs $\text{NTIME}[2^{O(m^d \log^d(Nm))}] = \text{NTIME}[2^{O(N^{d/r}\log^d N)}]$. To get all bits of an evaluation for a single element in H therefore takes at most $\text{NTIME}[O(m^e \log(Nm)) \cdot 2^{O(N^{d/r}\log^d N)}]$, which we can bound as $\text{NTIME}[2^{O(N^{d/r}\log^d N)}]$. To construct the entire set H we can use the same asymptotic time bound assuming wlog. that $d \geq 2$.

If during the process of obtaining all the bits we obtain a flag bit set to 0, we reject. This means that on every path where we pass this check, we have obtained a hitting set, unless N is an input length where the test property is not satisfied. On these paths, we continue to verify deterministically that $f(h) = 0$ for all $h \in H$. If yes, then we accept, else reject. By our previous remarks, for infinitely many N, this correctly decides whether $\Phi \equiv 0$.

Let us consider the cost of evaluation of Φ on elements of H. For $a \in S^\ell$ and subset S_j in the Nisan-Wigderson design, the bit size of $p_m(a_{|S_j})$ is $O(m^e \log(Nm))$. By Corollary 8 this means that the absolute value of any gate of Φ for input $NW^{p_m}(a)$ is at most $2^{O(N^2 m^{2e} \log^2(Nm))} = 2^{N^{O(1)}}$. Thus intermediate values can be represented by $poly(N)$ bits. We conclude that evaluation of Φ on a single element of the test set H cost time $poly(N)$. We can conclude the entire cost of our test algorithm is $\text{NTIME}[2^{O(N^{d/r}\log^d N)}]$. As r can be chosen arbitrarily large and d is an absolute constant not depending on r, we conclude that low-PIT $\in \bigcap_{\epsilon>0}$ i.o-NTIME$[2^{N^\epsilon}]$. ◀

5.1 Proof of Theorem 6

We first prove the hardness-to-randomness direction. The following corollary to Lemma 21 follows straightforwardly:

▶ **Corollary 23.** *Let $s(n) = n^{\omega(1)}$ be a function. Suppose that $\{p_n\}$ is a family of multilinear polynomials in n variables with coefficients of bit size at most $n^{e'}$ for some integer e', such that p_n cannot be written as q_n/c_n for $c_n \in \mathbb{Z}\backslash\{0\}$ for any q_n and c_n computed by constant-free arithmetic circuits of size $s(n)$. Then there exists an absolute constant $c > 0$ such that for any division-free constant-free arithmetic circuit Φ of size n with $deg(\Phi) \leq n$, if we take*

m such that $s(m) \cdot m^{-e'c} > n^c$ and let ℓ be given by Lemma 20, then for all large enough n, $\Phi \equiv 0 \Leftrightarrow (\forall a \in S^\ell), \Phi(NW^{p_m}(a)) = 0$, where $S = [nm+1]$.

We will describe an i.o-NTIME$[2^{n^{o(1)}}]/n^{o(1)}$ algorithm for low-PIT. Let Φ be an arithmetic circuit of size N, and let f be the polynomial computed by it. First we check that the formal degree of Φ is bounded by N, if not reject. Else, consider the given family $\{p_m\}$. By reindexing we may assume wlog. that p_m is defined over m variables. Let $e' \geq 1$ be such that p_m has coefficients of bit size at most $m^{e'}$. We have that for infinitely many m, p_m has ASIZEDEG′-hardness larger than $s(m)$, where $s(m) = m^{\omega(1)}$. The m that have this property we call *good*.

We use the complementation property for ml·NE/*lin*, cf. Proposition 3 and the comment thereafter. This means that we have a nondeterministic flag machine M running in time $2^{O(n')}$ with $O(n')$ bits of advice for inputs of size n' that can compute the characteristic function of $E(\{p_m\})$ Let c be the constant given by Corollary 23. For input size N the algorithm receives two strings of advice α and β. First, if there exists a good m_0 such that $s(m_0)(m_0)^{-ce'} \in [N^c, (N+1)^c]$, then $\alpha = 1^{m_0}$. If there is no such m_0, then α is set to the empty string. A simple argument shows that $|\alpha| = N^{o(1)}$. For the second piece of advice β we obtain the advice M needs so we can complete the simulations which we describe below (we will analyze this in more detail there).

In case the algorithm receives the empty string for α, it halts and rejects. Otherwise, we set $m = m_0$. Note that as N^c is a strict monotone increasing function it must be that for infinitely many N we obtain a good m_0 as advice. By Corollary 23, provided N is large enough, the following test property holds: $\Phi \equiv 0 \Leftrightarrow (\forall a \in S^\ell), \Phi(NW^{p_m}(a)) = 0$, where $S = [Nm+1]$ with $\ell = O(m^2/\log N)$ taken according to Lemma 20.

Let us continue the description of the algorithm. We produce a set H to sample Φ with, namely take H to be the output of $NW^{p_m}(\cdot)$ on S^ℓ. We have that $|H| = (Nm+1)^\ell = 2^{N^{o(1)}}$.

We do simulations of the machine M for $E(\{p_m\})$ to get all the bits of all the evaluations of $p_m(\cdot)$ on S^ℓ. As p_m is multilinear with coefficient of at most $m^{e'}$ many bits, we can bound the bit size of any such evaluation by $O(m^{e'} \log(Nm))$. This means that the inputs $(1^m, a_1, \ldots, a_m, i, b)$ that we simulate M on, have bit size $O(m \log(Nm))$ (recall i is given in binary). For the string β we give the advice that M needs for all input lengths up to this maximum bit size, which is $O(m^2 \log^2(Nm)) = N^{o(1)}$ in total. Given such advice, the simulation for a single such input thus costs NTIME$[2^{O(m \log(Nm))}]$ = NTIME$[2^{N^{o(1)}}]$. To get all bits of an evaluation for a single element in H therefore takes at most NTIME$[O(m^{e'} \log(Nm)) \cdot 2^{N^{o(1)}}]$ = NTIME$[2^{N^{o(1)}}]$ with the same amount of advice. We conclude that we can construct the entire set H in NTIME$[2^{N^{o(1)}}]$ with $N^{o(1)}$ advice.

If during the process of obtaining all the bits we obtain a flag bit set to 0, we reject. This means that if on every path where we pass this check, we have obtained a hitting set, provided N is large enough. On the path where we pass this check, we continue to verify deterministically that $f(h) = 0$ for all $h \in H$. If yes, then we accept, else reject. By our previous remarks, for infinitely many N, this correctly decides whether $\Phi \equiv 0$.

Let us consider the cost of evaluation of Φ on elements of H. For $a \in S^\ell$ and subset S_j in the Nisan-Wigderson design, the bit size of $p_m(a_{|S_j})$ by $O(m^{e'} \log(Nm))$. By Corollary 8 this means that the absolute value of any gate of Φ for input $NW^{p_m}(a)$ is at most $2^{O(N^2 m^{2e'} \log^2(Nm))} = 2^{N^{O(1)}}$. Thus intermediate values can be represented by $poly(N)$ bits. We conclude that evaluation of Φ on a single element of the test set H cost time $poly(N)$. We can conclude the entire cost of our test algorithm is $NTIME[2^{N^{o(1)}}]$ with $N^{o(1)}$ advice, and that for infinitely many input lengths N the algorithm is correctly decides low-PIT. ◀

Due to space restriction the randomness-to-hardness direction of the proof of Theorem 6 has been ommited from this version of the paper. It will appear in the full version.

References

1 S. Aaronson and A. Wigderson. Algebrization: A new barrier in complexity theory. *Transactions on Computation Theory*, 1(1), 2009.

2 L. Babai, L. Fortnow, and C. Lund. Nondeterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1:3–40, 1991. Addendum in vol. 2 of same journal.

3 P. Bürgisser. *Completeness and Reduction in Algebraic Complexity Theory*. Springer Verlag, 2000.

4 R. DeMillo and R. Lipton. A probabilistic remark on algebraic program testing. *Inf. Proc. Lett.*, 7:193–195, 1978.

5 D. van Melkebeek J. Kinne and R. Shaltiel. Pseudorandom generators, typically correct derandomization, and circuit lower bounds. Technical Report TR10–129, Electronic Colloquium on Computational Complexity (ECCC), 2010.

6 M. Jansen. Extracting roots of arithmetic circuits by adapting numerical methods. In *Proc. 2nd Symp. on Innovations in Computer Science*, 2011.

7 V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity testing means proving circuit lower bounds. *Computational Complexity*, 13(1–2):1–44, 2004.

8 P. Koiran. Shallow circuits with high powered inputs. In *Proc. 2nd Symp. on Innovations in Computer Science*, 2011.

9 P. Koiran and S. Perifel. Interpolation in Valiant's theory. *Computational Complexity*, 20(1):1–20, 2011.

10 N. Nisan and A. Wigderson. Hardness versus randomness. *J. Comp. Sys. Sci.*, 49:149–167, 1994.

11 R. Santhanam. Circuit lower bounds for Merlin–Arthur classes. *SIAM J. Comput.*, 39(3):1038–1061, 2009.

12 N. Saxena. Progress of polynomial identity testing. Technical Report ECCC TR09-101, Electronic Colloquium in Computational Complexity, 2009.

13 J.T. Schwartz. Fast probabilistic algorithms for polynomial identities. *J. Assn. Comp. Mach.*, 27:701–717, 1980.

14 V. Strassen. Vermeidung von divisionen. *Journal für die Reine und Angewandte Mathematik*, 264:182–202, 1973.

15 S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20:865–877, 1991.

16 L. Valiant. The complexity of computing the permanent. *Theor. Comp. Sci.*, 8:189–201, 1979.

17 R. Williams. Non-uniform ACC circuit lower bounds. In *Proceedings of 26th IEEE Conference on Computational Complexity*, page To appear, 2011.

18 R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Manipulation (EUROSAM '79)*, volume 72 of *Lect. Notes in Comp. Sci.*, pages 216–226. Springer Verlag, 1979.

Surface Split Decompositions and Subgraph Isomorphism in Graphs on Surfaces

Paul Bonsma[1]

1 **Humboldt University Berlin, Computer Science Department, Unter den Linden 6, 10099 Berlin, Germany.**
bonsma@informatik.hu-berlin.de.

Abstract

The Subgraph Isomorphism problem asks, given a *host graph* G on n vertices and a *pattern graph* P on k vertices, whether G contains a subgraph isomorphic to P. The restriction of this problem to planar graphs has often been considered. After a sequence of improvements, the current best algorithm for planar graphs is a linear time algorithm by Dorn (STACS '10), with complexity $2^{O(k)} \cdot O(n)$.

We generalize this result, by giving an algorithm of the same complexity for graphs that can be embedded in surfaces of bounded genus. In addition, we simplify the algorithm and analysis. The key to these improvements is the introduction of surface split decompositions for bounded genus graphs, which generalize sphere cut decompositions for planar graphs. We extend the algorithm for the problem of counting and generating all subgraphs isomorphic to P, even for the case where P is disconnected. This answers an open question by Eppstein (JGAA'99).

1998 ACM Subject Classification F.2.2 Computations on discrete structures, G.2.2 Graph algorithms

Keywords and phrases Analysis of algorithms, parameterized algorithms, graphs on surfaces, subgraph isomorphism, dynamic programming, branch decompositions, counting problems

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.531

1 Introduction

The Subgraph Isomorphism problem asks, given a *host graph* G on n vertices and a *pattern graph* P on k vertices, whether G contains a subgraph isomorphic to P. This is a well-studied problem that generalizes many other important problems, such as finding cliques, determining the girth, finding complete bipartite subgraphs, and finding a Hamilton path or cycle. See Eppstein [15] for a survey on previous results for this problem and its many applications. This problem is NP-complete in general, even for planar graphs. However, in many cases P can be considered to be a small fixed graph. In that case, a trivial polynomial time algorithm of complexity $n^{O(k)}$ exists. For general graphs, nothing better is known. When restricting G to be planar, this can be improved significantly: Eppstein [15] gave a linear time algorithm for Planar Subgraph Isomorphism for any fixed graph P on k vertices. This seems best possible. However to judge the practicality of such an algorithm, the dependency of the complexity on the value k is also essential. Hence we view the problem as a *parameterized problem* with parameter k. (See [14, 20] for background on parameterized algorithms.) Using this refined viewpoint, the complexity of Eppstein's algorithm [15] is $2^{O(k \log k)} \cdot O(n)$. This improved on previous algorithms for Planar Subgraph Isomorphism by Plehn and Voigt [21] of complexity $2^{O(k \log k)} \cdot n^{O(\sqrt{k})}$, and Alon et al. [3], of complexity $2^{O(k)} \cdot n^{O(\sqrt{k})}$. Finally, Dorn [11] improved the previous results and gave an algorithm of

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 531–542
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

complexity $2^{O(k)} \cdot O(n)$. Eppstein [16] also considered graphs of bounded genus, which generalize planar graphs. In [16], an algorithm for Subgraph Isomorphism in bounded genus graphs is given, with complexity $2^{O(k \log k)} \cdot O(n)$. In addition, Eppstein [16] considered the even more general graph class of apex-minor free graphs, and gave an $f(k) \cdot O(n)$ time algorithm, where $f(k)$ is a rapidly growing function of k. Subsequent results of Demaine and Hajiaghayi [7] imply that this complexity can be improved to $f(k) = 2^{O(k \log k)}$. The aforementioned results by Eppstein [15, 16] in fact hold for the more general *counting version* of the problem, where the number of subgraphs of G that are isomorphic to P should be computed. In addition, in the case P is connected, he gave an algorithm for *listing* all of these subgraphs in time $2^{O(k \log k)} \cdot O(n) + z \cdot k^{O(1)}$, where z is the number of such subgraphs.

New results In this paper, we give an algorithm of complexity $2^{O(k)} \cdot O(n)$ for the counting version of Subgraph Isomorphism, for the case where G has bounded genus. This generalizes the result for planar graphs in [11] and improves the complexity of the bounded genus result in [16]. In addition, we give an algorithm that lists all z subgraphs isomorphic to P in in time $2^{O(k)} \cdot O(n) + z \cdot k^{O(1)}$. This also holds for the case where P is disconnected, and therefore answers an open question from Eppstein [15]. This is made possible by developing a simpler method for counting disconnected subgraphs. Our results hold for graphs of bounded non-orientable genus as well. For simplicity, we describe the orientable case only, and discuss the non-orientable case in the full version of this paper [5]. We also remark that our algorithm can easily be modified to count *induced* subgraphs [5].

Related research There are many examples of problems that can be solved faster on planar graphs and generalizations such as bounded genus graphs and H-minor free graphs. For instance, for the aforementioned graph classes, many parameterized problems can be solved in *subexponential time* $2^{O(\sqrt{k})} \cdot O(n)$, see e.g. [2, 10, 6, 8]. The *theory of bidimensionality* [6, 8] easily gives subexponential time algorithms for many problems restricted to the aforementioned graph classes. However, Subgraph Isomorphism is not one of these, except for very special cases of P such as paths. An essential ingredient for many of these algorithmic results on planar graphs, bounded genus graphs and H-minor free graphs is *fast dynamic programming over tree decompositions and branch decompositions*. For many problems (such as for instance Longest Path, Maximum Leaf Tree), the best known dynamic programming algorithms for general graphs have a complexity of $2^{O(w \log w)} \cdot O(n)$, where w is the width of the given decomposition. However, when restricted to sparse graph classes, a lot of research has been devoted to showing that this can often be improved to $2^{O(w)} \cdot O(n)$ [1, 10, 11, 12, 13, 22]. In the case of planar graphs, an essential tool is given by a special kind of branch decompositions, called *sphere cut decompositions*. These were introduced by Seymour and Thomas [23], and their algorithmic usefulness was first demonstrated in 2005 (conference version) by Dorn et al. [13]. Loosely speaking, a branch decomposition for a graph G consists of a labeled tree T, and every edge $e \in E(T)$ partitions the edges of G into two graphs G_1 and G_2. In a sphere cut decomposition, for every $e \in E(T)$ a simple closed curve in the plane exists (a *noose*), that separates the plane into two regions, one containing G_1 and the other containing G_2. In the case that a sphere cut decomposition of width w is given, an improved complexity of $2^{O(w)} \cdot O(n)$ can be proved for many dynamic programming algorithms, by using the fact that solutions can cross the nooses in a limited number of ways, which can be encoded by *non-crossing partitions*; see [13, 10].

It is generally believed that many algorithms for planar graphs can be extended to graphs of bounded genus. However, in the past, making this step has always been an intricate task. For instance, Dorn et al. [12] consider the Hamilton Cycle problem and related problems on graphs of bounded genus, and reduce this case to the planar case by cutting the surface a

number of times along nooses. For the remaining planar case, dynamic programming over sphere cut decompositions is used, but this is relatively complex because the previous cuts need to be taken into account. Rué et al. [22] proposed a different dynamic programming method for graphs on surfaces: they define surface cut decompositions, where two subgraphs G_1 and G_2 defined by an edge of the branch decomposition are separated by a limited number of nooses, which have a limited number of common points. Since in the case of surfaces of higher genus, the boundary between G_1 and G_2 may become quite complex, an elaborate definition is required to characterize this [22]. After a few intermediate steps, the complexity bounds given in [22] are again based on counting non-crossing partitions.

New techniques and overview of the paper In Section 3 we give a dynamic programming algorithm for Subgraph Isomorphism that works for all graphs, when a branch decomposition is given. However, in order to give a good bound on its complexity, we restrict to bounded genus graphs and introduce a special kind of branch decomposition. *One of the main contributions of this paper is that we introduce surface split decompositions for graphs of bounded genus, and a strong but simple technique for bounding the resulting dynamic programming complexity.* This is a type of branch decomposition that directly generalizes sphere cut decompositions. It allows algorithms and analysis that are significantly simpler than previous dynamic programming algorithms for bounded genus graphs. In fact, our algorithm and analysis is even significantly simpler than that of some previous algorithms for planar graphs (e.g. [11]). Informally, our basic but crucial observation is that for surfaces of higher genus, it is irrelevant that the two subgraphs G_1 and G_2 defined by an edge of the branch decomposition can share a complex boundary; it is only relevant that there are two disjoint (connected) regions R_1 and R_2 in the surface such that G_i is drawn in R_i for $i = 1, 2$. This is because it is not necessary to consider the number of ways in which partial solutions can cross the boundary of the regions; we can argue on a higher level by appropriately applying a $2^{O(k)}$ bound on the total number of bounded genus graphs on k vertices, which avoids many of the technical challenges that were faced in previous papers for similar problems [1, 22, 11, 12]. For more details, see Section 4. In a subsequent paper, we will show how this technique can be applied to various other problems for graphs on surfaces.

In Section 5 we give an algorithm for finding surface split decompositions in linear time, where the width depends linearly on the graph diameter. For this we generalize a result by Tamaki [24] and Dorn [11], for finding low width sphere cut decompositions for planar graphs of bounded diameter. This is then applied in Section 6 to prove our main algorithmic results. To this end, we start with a standard technique for decomposing embedded graphs into *layers* of small branch width (see e.g. [2, 11, 15]). The main innovation in Section 6 is a new, simpler way (compared to [15]) of avoiding double-counting in the case where the pattern graph P is disconnected, by considering color-coded solutions.

We expect that the notion of surface split decompositions, our algorithm for finding them, and our technique for bounding the size of corresponding dynamic programming tables will be a stimulus for the algorithmic research on bounded genus graphs. We believe that it will enable the generalization of various existing algorithmic results for planar graphs, and that it should allow for the simplification of various known results for bounded genus graphs, and more general graphs. We remark that algorithms that are conceptually simpler are not only convenient for the reader or programmer; often they are also faster in practice. Indeed, when restricted to planar graphs, the constants in our complexity bound are significantly smaller than those in the algorithm by Dorn [11]. This is discussed in the full version of this paper [5]. The proofs and proof details that are omitted because of space constraints can also be found in [5]. We start by giving definitions in Section 2.

2 Preliminaries

For basic graph theoretical notations not defined here we refer to [9]. The main graphs that we will consider throughout will be *simple*, but we will construct auxiliary graphs that may have parallel edges and loops, i.e. that may be *multi-graphs*. The *distance* from u to v is the length of a shortest path from u to v. The *eccentricity* of a vertex $u \in V(G)$ is the maximum distance from u to v, over all $v \in V(G)$. By $d(v)$ we denote the *degree* of $v \in V(G)$, which is the number of incident edges. An *isomorphism* between two simple graphs G_1 and G_2 is a bijective function $\phi : V(G_1) \to V(G_2)$ such that $uv \in E(G_1)$ if and only if $\phi(u)\phi(v) \in E(G_2)$. The *Subgraph Isomorphism (counting) problem* takes as input a simple (host) graph G and a simple (pattern) graph P. The objective is to compute the number of subgraphs G' of G that are isomorphic to P. Such a subgraph G' is called a *P-isomorph*, or *isomorph* if the graph P is clear.

The set of leaves of a tree T is denoted by $L(T)$. A *branch decomposition* of a graph G is a tuple (T, μ) consisting of a ternary tree T, and a bijection $\mu : L(T) \to E(G)$. For a subset $S \subseteq L(T)$, we will use $\mu(S)$ to denote the set of images. Every edge $e_T \in E(T)$ defines a *middle set* $mid(e_T) \subseteq V(G)$ in the following way: let $T_1 \subseteq V(T)$ and $T_2 \subseteq V(T)$ denote the vertex sets of the two tree components of $T - e_T$. The edge sets $\mu(L(T) \cap T_1)$ and $\mu(L(T) \cap T_2)$ partition the edges of G. By G_1 and G_2 we denote the respective subgraphs of G induced by these edge sets. Now $\text{mid}(e_T)$ is defined as $V(G_1) \cap V(G_2)$. The *width* of (T, μ) is defined as $\max_{e_T \in E(T)} |\text{mid}(e_T)|$. A *rooted* branch decomposition is a tuple (T, μ) where T is a ternary tree, and a root $r \in L(T)$ is identified. In this case, μ is a bijection from $L(T) \backslash \{r\}$ to $E(G)$. Note that a rooted branch decomposition can easily be obtained from a branch decomposition. In the case of a rooted branch decomposition, for $e \in E(T)$, by $T_e \subseteq V(T)$ we denote the set of vertices of the component of $T - e$ that does not contain r. Similar to above, T_e defines a subgraph of G, which is denoted by G_e. Since T is ternary, every edge $e \in E(T)$ for which T_e is non-trivial (i.e. not a single vertex) has two *children*; these are the two edges of T_e that are adjacent to e. Observe that if $e_r \in E(T)$ is the edge incident with r, then $G_{e_r} = G$.

For an introduction to graphs embedded on surfaces we refer to [19]. Formally, a *surface* is a connected compact 2-manifold without boundary. For every integer $g \geq 0$, Let $\mathbb{S}_g$ denote a surface that is obtained by adding g handles to a sphere. Hence $\mathbb{S}_g$ is an orientable surface of genus g. For ease of presentation, all surfaces that we consider will be orientable. In the full version of this paper [5], we discuss the non-orientable case. A *region* of a surface is a connected open set. The *boundary* of a region R consists of all the points that lie in the closure of R but not in R itself. For a simple curve $C \subseteq \mathbb{S}_g$, all the points in C that are not the end points are called *interior points*.

An *embedding* ψ of a graph G into the surface $\mathbb{S}_g$ consists of an injective mapping of the vertices v of G to points $\psi(v)$ in $\mathbb{S}_g$, and a mapping of edges $e = uv$ of G to a simple curve $\psi(e)$ in $\mathbb{S}_g$ with end points $\psi(u)$ and $\psi(v)$, such that two edges may only intersect in their end points. To simplify terminology and notation, if ψ is an embedding of a graph G, then the images $\psi(v)$ and $\psi(e)$ (which are subsets of $\mathbb{S}_g$) for $v \in V(G)$ and $e \in E(G)$ will also be called *vertices* and *edges* of G, respectively. Let $X \subseteq \mathbb{S}_g$ be the union of all vertices and edges of an embedded graph G. The *faces* of G are the maximal regions of $\mathbb{S}_g \backslash X$. An embedding is a *2-cell embedding* if every face is homeomorphic to an open disc. For a graph G with n vertices and m edges, which is 2-cell embedded in $\mathbb{S}_g$ with f faces, by *Euler's formula* it holds that $n - m + f = 2 - 2g$. The boundary of every face F of G defines a closed walk (vertex sequence) $v_0, v_1, \ldots, v_{k-1}, v_0$ in G in a straightforward way, which will be called the

facial walk for F. If the length of W is k, then F will be called a *k-face*.

Given a 2-cell embedding ψ of a graph G in $\mathbb{S}_g$, we can define the following two related (multi-) graphs, and their 2-cell embeddings in $\mathbb{S}_g$. To construct the *dual graph* G^*, we draw one vertex v_F in every face F of G. For every edge $e \in E(G)$ that is incident with faces F and F', we draw an edge e^* in G^* from v_F to $v_{F'}$. e^* is called the *dual edge* of e and e the *primal edge* of e^*. The *radial graph* $\mathcal{R}_G$ of G (which is also called the *vertex-face incidence graph*) is obtained by starting with the vertex set $V(G)$ (the *original vertices*), and adding a vertex v_F for every face F of G (the *face vertices*). For every face F in G with corresponding facial walk $u_0, u_1, \ldots, u_{k-1}, u_0$, add k edges u_0v_F, u_1v_F, $\ldots$, $u_{k-1}v_F$ to $\mathcal{R}_G$, drawn in the region F, in the given order around v_F.

A *combinatorial embedding* π of a graph G consists of a cyclic order π_v on the incident edges for every vertex $v \in V(G)$. From an embedding of a graph we obtain the corresponding combinatorial embedding by considering the clockwise order of edges around every vertex. A *map* is a connected graph G together with a combinatorial embedding π. The set of all facial walks and therefore the number of faces of a map can easily be deduced (without constructing the embedding), so the number of faces f of a map is well-defined. The *genus* g of a map G, π is the solution to $n - m + f = 2 - 2g$, where $n = |V(G)|$, $m = |E(G)|$ and f is the number of faces of G, π. The *genus of a connected graph* G is the minimum g such that G admits a combinatorial embedding of genus g. Given a map G, π of genus g, a corresponding 2-cell embedding of G in $\mathbb{S}_g$ exists.

3 A dynamic program for counting colorful isomorphs

In this section we give a dynamic programming algorithm for the following generalization of Subgraph Isomorphism. An instance of the *Colorful Subgraph Isomorphism* problem consists of a colored host graph G, and a pattern graph P. The *coloring* of G is a function $\alpha : V(G) \to C$, with $C = \{1, \ldots, q\}$. This encodes a partition of $V(G)$ into q sets. We remark that this is not required to be a *proper* vertex coloring, so adjacent vertices may receive the same color. A subgraph G' of G is called *colorful* if for every color $x \in C$, G' contains a vertex v of color x. (Note that G' may have more than q vertices.) The objective is to count the number of colorful P-isomorphs of G. We now present an algorithm for this problem. (When $q = 1$, this is the original counting problem.)

Dynamic programming table Let (T, μ) be a rooted branch decomposition of G. For every edge $e \in E(T)$, we will form a dynamic programming table $\mathcal{T}_e$. Informally, this table will store information about all possible subgraphs of the graph G_e, on at most k vertices. Firstly, we distinguish between non-isomorphic subgraphs. Furthermore, subgraphs of G_e that are isomorphic but intersect differently with the 'boundary' $\mathrm{mid}(e)$ of G_e are also considered distinct. Finally, we keep track of the set of colors that appear in these subgraphs. Two subgraphs of G_e are only considered equivalent if they match in all three regards. In that case, there will be a single entry in the table that represents both subgraphs. We now define this formally.

Let H be a graph, and let γ be a mapping from $\mathrm{mid}(e)$ to $V(H) \cup \{\mathrm{nil}\}$, which is *injective on* $V(H)$. To be precise, every vertex of $V(H)$ occurs at most once as a γ-image, but multiple vertices may be mapped to nil. Furthermore, let $A \subseteq \{1, \ldots, q\}$. For such a tuple (H, γ, A), a subgraph G' of G_e is called an *(H, γ, A)-subgraph* if the following two properties hold.

- There is an isomorphism $\phi : V(G') \to V(H)$ with $\gamma(v) = \phi(v)$ for all $v \in \mathrm{mid}(e) \cap V(G')$, and $\gamma(v) = \mathrm{nil}$ for all $v \in \mathrm{mid}(e) \setminus V(G')$.
- For all colors $x \in C$: $x \in A$ if and only if G' contains a vertex of color x.

For $e \in E(T)$, the *dynamic programming table* $\mathcal{T}_e$ will now contain *entries* (H, γ, A, η), where H, γ and A are as defined above, and η is a non-negative integer. The idea is that such a table entry indicates that G_e contains exactly η non-equivalent (H, γ, A)-subgraphs. Table entries $(H_1, \gamma_1, A_1, \eta_1)$ and $(H_2, \gamma_2, A_2, \eta_2)$ are *equivalent* if the following properties hold.

- There is an isomorphism $\phi : V(H_1) \to V(H_2)$ such that for all $v \in \text{mid}(e)$, either $\gamma_1(v) = \gamma_2(v) = \text{nil}$, or $\phi(\gamma_1(v)) = \gamma_2(v)$ holds.
- $A_1 = A_2$.

Observe that the above definition satisfies the following property: a subgraph G' of G_e is both an (H_1, γ_1, A_1)-subgraph and an (H_2, γ_2, A_2)-subgraph if and only if $(H_1, \gamma_1, A_1, \eta_1)$ and $(H_2, \gamma_2, A_2, \eta_2)$ are equivalent. Therefore, when two entries $(H_1, \gamma_1, A_1, \eta_1)$ and $(H_2, \gamma_2, A_2, \eta_2)$ are equivalent, we can *merge* them by replacing them with the single entry $(H_1, \gamma_1, A_1, \eta_1 + \eta_2)$. We say that the table $\mathcal{T}_e$ is *k-correct* if

- for every tuple (H, γ, A), $\mathcal{T}_e$ contains an entry (H, γ, A, η) if and only if G_e contains exactly $\eta \geq 1$ graphs G' with $|V(G')| \leq k$ that are (H, γ, A)-subgraphs, and
- $\mathcal{T}_e$ contains no pairs of equivalent entries.

Dynamic programming update step Let $e \in E(T)$ be an edge with children f and g. We will now show how a k-correct table $\mathcal{T}_e$ for e can be obtained from k-correct tables $\mathcal{T}_f$ and $\mathcal{T}_g$ for f and g, respectively. We define two entries $(H_f, \gamma_f, A_f, \eta_f) \in \mathcal{T}_f$ and $(H_g, \gamma_g, A_g, \eta_g) \in \mathcal{T}_g$ to be *compatible* if for all $v \in \text{mid}(f) \cap \text{mid}(g)$, it holds that $\gamma_f(v) = \text{nil}$ if and only if $\gamma_g(v) = \text{nil}$. We now show how to *combine* two such compatible entries into an entry $(H_e, \gamma_e, A_e, \eta_e)$:

- For all $v \in \text{mid}(f) \cap \text{mid}(g)$ with $\gamma_f(v) \neq \text{nil}$ (and thus $\gamma_g(v) \neq \text{nil}$): identify the vertex $\gamma_f(v)$ of H_f with the vertex $\gamma_g(v)$ of H_g, and call the new vertex $\nu(v)$. This gives the graph H_e.
- For all $v \in \text{mid}(e)$: If $v \in \text{mid}(f) \backslash \text{mid}(g)$ then set $\gamma_e(v) = \gamma_f(v)$. If $v \in \text{mid}(g) \backslash \text{mid}(f)$ then set $\gamma_e(v) = \gamma_g(v)$. If $v \in \text{mid}(g) \cap \text{mid}(f)$ then set $\gamma_e(v) = \nu(v)$. By definition of $\text{mid}(e)$, this covers all cases and thus defines the function γ_e.
- Set $A_e = A_f \cup A_g$.
- Set $\eta_e := \eta_f \cdot \eta_g$.

It can be verified that if there are η_f (H_f, γ_f, A_f)-subgraphs in G_f, and η_g (H_g, γ_g, A_g)-subgraphs in G_g, then there are $\eta_f \cdot \eta_g$ (H_e, γ_e, A_e)-subgraphs in G_e that are the result of combining graphs of the former two types in every possible combination. However, there may also be (H_e, γ_e, A_e)-subgraphs of G_e that are the result of combining different types of subgraphs of G_f and G_g. Therefore, merging entries is required as well.

Assuming that we have k-correct tables $\mathcal{T}_f$ and $\mathcal{T}_g$ for f and g respectively, we construct a k-correct table $\mathcal{T}_e$ for e as follows: We start with an empty table $\mathcal{T}_e$. Then we consider every pair of compatible entries from $\mathcal{T}_f$ and $\mathcal{T}_g$, and combine them as described above. For every such combination, this yields a possible entry (H, γ, A, η) for $\mathcal{T}_e$. In case that H has more than k vertices, we ignore this possible entry. Otherwise, we check whether $\mathcal{T}_e$ already contains an equivalent entry (H', γ', A', η'). If so, we merge the two entries. If not, we add the entry (T, γ, A, η) to the table $\mathcal{T}_e$. Then we continue with the next pair of compatible entries from $\mathcal{T}_f$ and $\mathcal{T}_g$. This yields the following lemma.

▶ **Lemma 1.** *Let (T, μ) be a rooted branch decomposition for a colored graph G, and k be an integer. Let $e \in E(T)$ be an edge with children f and g, for which k-correct tables $\mathcal{T}_f$ and $\mathcal{T}_g$ are given. Then the table $\mathcal{T}_e$ that is constructed with the above dynamic programming update step is k-correct for e. The construction takes time $X^3 \cdot f(k) \cdot k^{O(1)}$, where $f(k)$ is the complexity of deciding whether two entries are equivalent, and X is an upper bound on the size of a k-correct table.*

This update step is the core of the following dynamic programming algorithm: First, for every edge $e \in E(T)$ that has no children, G_e consists of a single edge, so it is trivial to construct a k-correct table $\mathcal{T}_e$. For every edge $e \in E(T)$ with two children f and g, we compute a k-correct table $\mathcal{T}_e$ using the dynamic programming update step. After computing k-correct tables for all edges of T, we inspect the table $\mathcal{T}_{e_r}$ where e_r is the root edge of (T, μ). Since $G_{e_r} = G$, $\text{mid}(e_r) = \emptyset$. Therefore, $\mathcal{T}_{e_r}$ contains at most one entry (H, γ, A, η) such that H is isomorphic to P and $A = \{1, \ldots, q\}$. If there is such an entry (H, γ, A, η), we return η, and otherwise we return 0. The correctness of this procedure follows from the definitions. Combined with Lemma 1 this gives the following theorem. Note that a branch decomposition (T, μ) of a graph on m edges has $|E(T)| \in O(m)$.

▶ **Theorem 2.** *Let (T, μ) be a rooted branch decomposition of a colored graph G with m edges. In time $X^3 \cdot f(k) \cdot k^{O(1)} \cdot O(m)$, it can be computed how many colorful subgraphs of G are isomorphic to a given graph P on k vertices, where $f(k)$ is the complexity of deciding whether two entries are equivalent, and X is an upper bound on the size of a k-correct table.*

The above theorem applies to general graphs, for which an appropriate bound for X can be given (this way we would match Eppstein's result [15]). However, to obtain the desired improved complexity, we consider bounded genus graphs and introduce surface split decompositions in the next section. In the case that an embedding π of G of bounded genus is given, and (T, μ) is a surface split decomposition, we give a good upper bound for X.

4 Surface split decompositions and a bound for the table size

In this section, we introduce surface split decompositions for graphs embedded in $\mathbb{S}_g$, and demonstrate their usefulness. For graphs of genus 0 (planar graphs), the following definition is an alternative way to define sphere cut decompositions.

▶ **Definition 3.** Let G be a graph embedded in $\mathbb{S}_g$. A branch decomposition (T, μ) of G is called a *surface split decomposition* if for every $e \in E(T)$ and corresponding subgraphs G_1 and G_2 of G, there are disjoint regions $R_1 \subseteq \mathbb{S}_g$ and $R_2 \subseteq \mathbb{S}_g$ such that for $i = 1, 2$, all vertices and edges of G_i are drawn entirely in the closure of R_i.

Observe that this definition implies that all vertices in $\text{mid}(e)$ lie on the boundary of the closure of R_1 and on the boundary of the closure of R_2, so the closures are *not* disjoint (if $\text{mid}(e) \neq \emptyset$). We stress that it is crucial that R_1 and R_2 are connected *open* sets. If not, then firstly the above definition is not a generalization of sphere cut decompositions, but more importantly, the proof of Lemma 5 below fails. Even if the boundaries of the regions R_1 and R_2 are the same, this boundary is not necessarily a simple curve if $g \geq 1$. It may even be quite complex [22], but this does not matter. The definition easily extends to *maps* G, π of genus g: (T, μ) is a *surface split decomposition* of G, π if it is a surface split decomposition for any embedding of G in $\mathbb{S}_g$ that corresponds to the combinatorial embedding π.

Let (T, μ) be a surface split decomposition for a map G, π. We now give a bound on the size of a k-correct table. We will use a bound on the number of different graphs on n vertices, embedded in a surface of genus g. To be precise, we will consider *simple, connected* graphs G, that come with a combinatorial embedding π. Such a pair G, π is called a *simple map*. In addition, a tuple (u, v) of vertices is given such that $uv \in E(G)$. This is the *root*. Such a combination $G, \pi, (u, v)$ is called a *simple rooted map*. Two simple rooted maps $G, \pi, (u, v)$ and $G', \pi', (u', v')$ are *equivalent* if there is an isomorphism $\phi : V(G) \to V(G')$ with $\phi(u) = u'$, $\phi(v) = v'$, and that maps facial walks of G to facial walks of G'. In case

an edge labeling is given for both simple rooted maps, we also require that ϕ maps edges to edges of the same label. To bound the number of simple rooted maps, we apply a result by Bender and Canfield [4], which implies the following bound.

▶ **Theorem 4.** *There are $2^{O(n)} \cdot n^{O(g)}$ simple rooted maps of genus at most g on at most n vertices.*

▶ **Lemma 5.** *Let G, π be a simple map of genus at most g, for which a coloring with q colors and a surface split decomposition (T, μ) of width w are given. Let k be an integer. For $e \in E(T)$, let $\mathcal{T}_e$ be a k-correct table. Then $|\mathcal{T}_e| \in 2^q \cdot 2^w \cdot 2^{O(k)} \cdot k^{O(g)}$.*

Proof sketch: For all entries $(H, \gamma, A, \eta) \in \mathcal{T}_e$ where H has at most k vertices, we will *encode* H and γ by a simple rooted map H', π', (u, v) on at most $k+1$ vertices, together with a 0/1-labeling λ of a subset of the edges of H', and a 0/1-labeling ρ of the vertices in $\mathrm{mid}(e)$. This means that any two non-equivalent entries (H, γ, A, η) and (H', γ', A', η') either have $A \neq A'$, or yield a different labeling ρ, or yield non-equivalent rooted maps.

We use the following auxiliary graph. Since (T, μ) is a surface split decomposition, there are disjoint regions R_1 and R_2 in $\mathbb{S}_g$, such that G_e lies in the closure of R_1, and $\mathrm{mid}(e)$ lies on the boundary of both R_1 and R_2. Thus we can extend G_e by drawing a vertex u in R_2, and drawing edges in the closure of R_2 from u to every vertex in $\mathrm{mid}(e)$, while maintaining an embedding in $\mathbb{S}_g$. Number the vertices of $\mathrm{mid}(e)$ $v_1, \ldots, v_t$, corresponding to the clockwise order of edges around u.

Now we show how to construct the encoding H', π', ρ, λ for an entry $(H, \gamma, A, \eta) \in \mathcal{T}_e$. Firstly, for all vertices $v \in \mathrm{mid}(e)$, set $\rho(v) = 0$ if and only if $\gamma(v) = \mathrm{nil}$. The graph H' is constructed as follows. Since $\mathcal{T}_e$ is k-correct, H corresponds to a subgraph of G_e, so H can also be drawn in the closure of R_1, such that all vertices that are γ-images are drawn on the boundary of R_1. Start with such an embedding. Next, add the vertex u, and edges uv_i for every $v_i \in \mathrm{mid}(e)$ with $\gamma(v_i) \neq \mathrm{nil}$. Draw these as described in the previous paragraph. This yields a simple graph embedded in $\mathbb{S}_g$, on at most $k+1$ vertices. However, it may not be connected. Add edges between different components until the graph is connected. This yields H'. Clearly, drawing these new edges can be done while maintaining an embedding. Hence H' is embedded in $\mathbb{S}_g$, and the corresponding combinatorial embedding π' has genus at most g. To obtain a *rooted map*, choose i to be the lowest index such that $\rho(v_i) = 1$. Then the tuple (u, v_i) is chosen as the root of H', π'. That is, $v = v_i$. A *bridge* of a connected graph G is an edge $e \in E(G)$ such that $G - e$ is disconnected. For all bridges $e \in E(H')$, we set $\lambda(e) = 1$ if $e \in E(H)$, and $\lambda(e) = 0$ otherwise. It can now be verified that the rooted edge labeled map $H', \pi', (u, v)$, λ, function ρ and set A encode the entry; informally, when knowing $H', \pi', (u, v)$, λ and ρ, we can reconstruct H and γ.

There are at most 2^w possibilities for ρ, at most 2^q possibilities for A, at most $2^{O(k)} \cdot k^{O(g)}$ simple rooted maps on at most $k+1$ vertices of genus at most g (Theorem 4), and at most 2^k possible labelings λ (since a graph on $k+1$ vertices contains at most k bridges.) Therefore, the number of entries in a k-correct table is bounded by $2^q \cdot 2^w \cdot 2^{O(k)} \cdot k^{O(g)}$. □

Using the isomorphism testing algorithm for graphs of bounded genus by Miller [18] or Grohe [17], we can test in time $k^{O(g)}$ whether two entries are equivalent. Combining this fact with Theorem 2 and Lemma 5 gives the following theorem.

▶ **Theorem 6.** *Let G, π be a simple map with m edges of genus at most g, for which a coloring with q colors and a surface split decomposition (T, μ) of width w are given. Let P be a graph on k vertices. In time $8^q \cdot 8^w \cdot 2^{O(k)} \cdot k^{O(g)} \cdot O(m)$, it can be computed how many colorful subgraphs of G are isomorphic to P.*

5 Constructing surface split decompositions

Tamaki [24] gave a linear time algorithm for constructing a branch decomposition of width $2d+1$ of a graph G embedded in the sphere, when a vertex $r \in V(G)$ of eccentricity d is given. Dorn [11] gave a different presentation of the construction and showed that it yields in fact a sphere cut decomposition. We now generalize this result to surfaces of higher genus.

▶ **Theorem 7.** *Let G, π be a map of genus at most g, for which a vertex $r \in V(G)$ of eccentricity d is given. In linear time, a surface split decomposition (T, μ) of G of width at most $(2g+1)(4d+3)/2$ can be constructed.*

Before we construct the surface split decomposition (T, μ), we will construct a number of auxiliary graphs. The objective of the first stage of the construction is to construct a tree T^*, such that for every $e \in E(G)$, there is a unique vertex of T^* associated with e. This can be thought of as the function μ, from a subset of $V(T^*)$ to $E(G)$. We will show in Lemmas 8 and 9 below that this T^* is already 'almost' the desired surface split decomposition. However, T^* will not yet be ternary (though it will have maximum degree 3), and the vertices that are associated with edges of G are not necessarily leaves. This is subsequently addressed in the second stage of the construction.

For the *first stage*, we start by modifying the graph G as follows: for every edge $e = uv$, add two extra parallel edges between u and v, one on either side of e (while maintaining a combinatorial embedding). This ensures that all original edges of G are incident with two 2-faces, and that every vertex has degree at least 3. Denote the resulting embedded graph by G'. The edges in $E(G') \cap E(G)$ are called *original edges*, and edges in $E(G') \backslash E(G)$ are called *new edges*. Now construct $\mathcal{R}_{G'}$, the radial graph of G'. Let T_S be a BFS spanning tree of $\mathcal{R}_{G'}$, rooted at a vertex $r \in V(\mathcal{R}_{G'}) \cap V(G)$ of eccentricity d. Choose T^* to be a spanning tree of the dual graph of $\mathcal{R}_{G'}$, such that for all edges $e^* \in E(T^*)$, the corresponding primal edge e is not in T_S. Using Euler's formula it can be shown that there are now $2g$ edges of $\mathcal{R}_{G'}$ that are neither in T_S, nor are their dual edges in T^*. Add all of these edges to T_S, to obtain the graph T_S^+. This completes the first stage of the construction of a surface split decomposition.

We remark that there is a trivial bijection between the faces of $\mathcal{R}_{G'}$ and the edges of G', and a trivial bijection between the vertices of T^* and the faces of $\mathcal{R}_{G'}$, and therefore a resulting bijection from $V(T^*)$ to $E(G')$. Below, we refer to these bijections when speaking of 'the set of edges of G' or faces of $\mathcal{R}_{G'}$ that *corresponds to* a subset of $V(T^*)$'. The *subgraph of G' that corresponds to a set $T_1 \subseteq V(T^*)$* is the subgraph of G' induced by the edges that correspond to T_1. For an edge $e_T \in E(T^*)$, let T_1 and T_2 be the vertex sets of the two components of $T^* - e_T$. Informally, by joining the faces of $\mathcal{R}_{G'}$ that correspond to the vertices in T_1 and T_2, we can construct regions $R(T_1)$ and $R(T_2)$ that satisfy the following two properties.

▶ **Lemma 8.** *For an edge $e_T \in E(T^*)$, let T_1 and T_2 be the vertex sets of the two components of $T^* - e_T$, and let G'_1 and G'_2 be the subgraphs of G' that correspond to T_1 and T_2 respectively. For every such edge e_T, there exist disjoint regions $R(T_1) \subseteq \mathbb{S}_g$ and $R(T_2) \subseteq \mathbb{S}_g$, such that for $i = 1, 2$, all vertices and edges of G_i are drawn entirely in the closure of R_i.*

▶ **Lemma 9.** *For an edge $e_T \in E(T^*)$, let T_1 be the vertex set of one of the components of $T^* - e_T$, and let G'_1 be the subgraph of G' that corresponds to T_1. For every such edge e_T, there are at most $(2g+1)(4d+3)/2$ vertices of G'_1 that lie on the boundary of $R(T_1)$.*

Proof sketch: We use that the boundary of a region $R(T_1)$ is a subgraph of $\mathcal{R}_{G'}$, and contains at most $2g+1$ edges that are not in T_S. This holds because one of these edges is the primal

edge of e_T, and the other edges are edges of $E(T_S^+)\backslash E(T_S)$, of which there are at most $2g$ (which can be shown using Euler's formula). Secondly, one can show that the maximum length of a path in T_S is $4d+2$. Hence the boundary of the region $R(T_1)$ contains at most $(2g+1)(4d+3)$ edges. This number may be divided by 2 since $\mathcal{R}_{G'}$ is bipartite and only half of its vertices are vertices of G'. □

At this point, T^* is already close to a low-width surface split decomposition of G. It remains to make it into a ternary tree, of which only the leaves correspond to original edges of G'. This is easily done with elementary operations. The reason is that, because of the parallel edges that were added to G', a vertex $v_d \in V(T^*)$ has degree at most 2 if it corresponds to an original edge of G', and degree at most 3 otherwise. We omit the details of this final stage of the proof of Theorem 7.

6 Summary of the algorithm for bounded genus graphs

In this section we show how to combine the ingredients of the previous sections to obtain our main results; we present an algorithm for counting the number of subgraphs isomorphic to P in a graph G of bounded genus, and an algorithm for listing all of them. Without loss of generality, we may assume that G is connected. Throughout this section we use the following notations. We choose an arbitrary vertex $r \in V(G)$. Let d be the eccentricity of r. For $i = 0, \ldots, d$, define $L_i \subseteq V(G)$ to be the set of vertices at distance i of r. We call such a set a *layer*. Let $G_i^j = G[L_i \cup L_{i+1} \cup \ldots \cup L_j]$. The following property easily follows from Theorem 7 using a standard argument, see e.g. [15, 11].

▶ **Lemma 10.** *Let G, π be a map of genus g, with a vertex r of eccentricity d, and subgraphs G_i^j as defined above. For i, j with $0 \leq i \leq j \leq d$, a surface split decomposition of G_i^j of width at most $(2g+1)(4j-4i+7)/2$ can be constructed in time $O(n'+m')$, where $n' = |V(G_i^j)|$ and $m' = |E(G_i^j)|$.*

We now present a novel algorithm for counting the number of P-isomorphs in G, even for the case where P is disconnected. Let c be the number of components of P, which are denoted by $P^1, \ldots, P^c$. For $S \subseteq \{1, \ldots, c\}$, denote by P^S the subgraph of P induced by the components with labels in S. Formally, $P^S = P[\cup_{i \in S} V(P^i)]$. For every G_i^j, we consider the following coloring, which uses the color set $\{1, \ldots, j-i+1\}$: the vertices of layer L_x are all colored with color $x-i+1$. For $0 \leq i \leq j \leq d$ with $j-i < k$ and $S \subseteq \{1, \ldots, c\}$, we define $\mathrm{DPC}_i^j(S)$ to be the number of *colorful* subgraphs of G_i^j that are isomorphic to P^S. In other words, this is the number of P^S-isomorphs that use all layers of G_i^j. Combining Lemma 10 with Theorem 6 yields the following proposition.

▶ **Proposition 11.** *Let G, π be a connected simple map of genus at most g, let P be a graph on k vertices with c components, and let $DPC_i^j(S)$ and G_i^j be as defined above. For given i, j with $0 \leq i \leq j \leq d$ and $j-i < k$, and given set $S \subseteq \{1, \ldots, c\}$, $DPC_i^j(S)$ can be computed in time $2^{O(gk)} \cdot O(n'+m')$, where $n' = |V(G_i^j)|$ and $m' = |E(G_i^j)|$.*

For $j \in \{0, \ldots, d\}$, we define $\mathrm{DPT}^j(S)$ to be the number of subgraphs of G_0^j that are isomorphic to P^S. Note that these are not required to be colorful. In particular, if $S = \emptyset$, then we define $\mathrm{DPT}^j(S) = 1$. To avoid the discussion of trivial cases, we simply define $\mathrm{DPC}_i^j(S) = 0$ if $i < 0$, and $\mathrm{DPT}^j(S) = 0$ if $j < 0$. It can be shown that the following recursion for computing $\mathrm{DPT}^j(S)$ is correct.

▶ **Lemma 12.** *For every $j \in \{0,\dots,d\}$ and $S \subseteq \{1,\dots,c\}$, it holds that $DPT^j(S) = DPT^{j-1}(S) + \sum DPT^{j-x-1}(S_1) \cdot DPC^j_{j-x+1}(S_2)$, where the summation is over all partitions of S into S_1 and S_2 with $S_2 \neq \emptyset$, and all $x \in \{1,\dots,k\}$.*

▶ **Theorem 13.** *Let G,π be a simple map with m edges, of genus at most g, and let P be a (possibly disconnected) graph on k vertices. In time $2^{O(gk)} \cdot O(m)$, the number of subgraphs of G that are isomorphic to P can be computed.*

Proof: In the *first stage* of the algorithm, we compute $\mathrm{DPC}^j_i(S)$ for every i, j with $0 \leq i \leq j \leq d$ and $j - i < k$, and every $S \subseteq \{1,\dots,c\}$. There are 2^c choices of S, so computing DPC^j_i for one choice of i and j and all possible choices of S takes time $2^c \cdot 2^{O(gk)} \cdot O(n'+m')$, where $n' = |V(G^j_i)|$ and $m' = |E(G^j_i)|$ (Proposition 11). Since every vertex and every edge of G appears in G^j_i for at most $O(k^2)$ choices of i and j, this yields a total complexity of $2^c \cdot 2^{O(gk)} \cdot k^2 \cdot O(n+m) \subseteq 2^{O(gk)} \cdot O(m)$ for the first stage. (G is connected, so $n \in O(m)$.) The *second stage* of the algorithm uses the recursion from Lemma 12 for computing $\mathrm{DPT}^j(S)$ for every $j \in \{0,\dots,d\}$ and $S \subseteq \{1,\dots,c\}$. One can show that this takes time $3^c \cdot O(k) \cdot O(d) \subseteq 3^c \cdot O(k) \cdot O(n) \subseteq 2^{O(k)} \cdot O(m)$. Finally, the algorithm returns the value of $\mathrm{DPT}^d(\{1,\dots,c\})$, which is the total number of P-isomorphs of $G^d_0 = G$. □

▶ **Corollary 14.** *Let g be a constant. Let G,π be a simple map with m edges, of genus at most g, and let P be a (possibly disconnected) graph on k vertices. In time $2^{O(k)} \cdot O(n)$, the number of subgraphs of G that are isomorphic to P can be computed.*

(Here we used $m \in O(n+g) = O(n)$.) Since the above algorithm is a rather simple algorithm that uses only additions and multiplications for the counting, it can be extended to an algorithm for listing all isomorphs. We sketch how this can be done for the second stage of the algorithm. We first run the above counting algorithm and compute all values $\mathrm{DPT}^j(S)$ and $\mathrm{DPC}^j_i(S)$. Next, we apply a backtracking stage where we recursively mark the combinations of i, j and S that contribute to the total number of isomorphs. In a third stage, we apply the counting algorithm again, but instead of computing the values for $\mathrm{DPT}^j(S)$ and $\mathrm{DPC}^j_i(S)$, we compute a *list* of all corresponding subgraphs of G for all combinations of i, j and S that actually contribute to the total number of isomorphs. The steps of the algorithm where lists are constructed can then be attributed to the construction of the final list of P-isomorphs, and thus their complexity can be bounded by $zk^{O(1)}$, where z is the number of P-isomorphs. All other steps can be done with the same complexity as the counting algorithm. This yields:

▶ **Theorem 15.** *Let G,π be a simple map with m edges, of genus at most g, and let P be a (possibly disconnected) graph on k vertices. In time $2^{O(gk)} \cdot O(m) + zk^{O(1)}$, all subgraphs of G that are isomorphic to P can be generated, where z is the number of such subgraphs.*

Acknowledgement The author would like to thank Frederic Dorn for the introduction to the subject and the many inspiring discussions.

References

1 I. Adler, F. Dorn, F. Fomin, I. Sau, and D. Thilikos. Fast minor testing in planar graphs. In *ESA '10*, volume 6346 of *LNCS*, pages 97–109. Springer, Berlin, 2010.

2 J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: exponential speed-up for planar graph problems. *J. Algorithms*, 52(1):26–56, 2004.

3 N. Alon, R. Yuster, and U. Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

4 E. A. Bender and E. R. Canfield. The asymptotic number of rooted maps on a surface. *J. Combin. Theory Ser. A*, 43(2):244–257, 1986.

5 P. Bonsma. Surface split decompositions and subgraph isomorphism in graphs on surfaces. eprint arXiv:1109.4554, 2011.

6 E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

7 E. D. Demaine and M. Hajiaghayi. Equivalence of local treewidth and linear local treewidth and its algorithmic applications. In *SODA '04*, pages 840–849. SIAM, Philadelphia, PA, USA, 2004.

8 E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

9 R. Diestel. *Graph theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, Heidelberg, fourth edition, 2010.

10 F. Dorn. *Designing subexponential algorithms: problems, techniques and structures*. PhD thesis, University of Bergen, Norway, 2007.

11 F. Dorn. Planar subgraph isomorphism revisited. In *STACS '10*, volume 5 of *LIPIcs*, pages 263–274. Dagstuhl Publishing, Dagstuhl, Germany, 2010.

12 F. Dorn, F. V. Fomin, and D. M. Thilikos. Fast subexponential algorithm for non-local problems on graphs of bounded genus. In *Algorithm theory—SWAT '06*, volume 4059 of *LNCS*, pages 172–183. Springer, Berlin, 2006.

13 F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin. Efficient exact algorithms on planar graphs: exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010.

14 R. G. Downey and M. R. Fellows. *Parameterized complexity*. Springer-Verlag, New York, 1999.

15 D. Eppstein. Subgraph isomorphism in planar graphs and related problems. *J. Graph Algorithms Appl.*, 3(3):1–27, 1999.

16 D. Eppstein. Diameter and treewidth in minor-closed graph families. *Algorithmica*, 27(3-4):275–291, 2000.

17 M. Grohe. Isomorphism testing for embeddable graphs through definability. In *STOC'00*, pages 63–72. ACM, New York, 2000.

18 G. Miller. Isomorphism testing for graphs of bounded genus. In *STOC'80*, pages 225–235. ACM, New York, 1980.

19 B. Mohar and C. Thomassen. *Graphs on surfaces*. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore, MD, 2001.

20 R. Niedermeier. *Invitation to fixed-parameter algorithms*, volume 31 of *Oxford Lecture Series in Mathematics and its Applications*. Oxford University Press, Oxford, 2006.

21 J. Plehn and B. Voigt. Finding minimally weighted subgraphs. In *WG '90*, volume 484 of *LNCS*, pages 18–29. Springer, Berlin, 1991.

22 J. Rué, I. Sau, and D. Thilikos. Dynamic programming for graphs on surfaces. In *ICALP '10*, volume 6198 of *LNCS*, pages 372–383. Springer Berlin / Heidelberg, 2010.

23 P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.

24 H. Tamaki. A linear time heuristic for the branch-decomposition of planar graphs. In *ESA '03*, volume 2832 of *LNCS*, pages 765–775. Springer Berlin / Heidelberg, 2003.

The Denjoy alternative for computable functions

Laurent Bienvenu[1], Rupert Hölzl[*2], Joseph S. Miller[†3], and André Nies[4]

1,2 LIAFA, CNRS & Université de Paris 7,
Case 7014, 75205 Paris Cedex 13, France
laurent.bienvenu@liafa.jussieu.fr, r@hoelzl.fr

3 Department of Mathematics, University of Wisconsin,
Madison, WI 53706-1388, USA
jmiller@math.wisc.edu

4 Department of Computer Science, University of Auckland,
Private Bag 92019, Auckland, New Zealand
andre@cs.auckland.ac.nz

Abstract

The Denjoy-Young-Saks Theorem from classical analysis states that for an arbitrary function $f\colon \mathbb{R} \to \mathbb{R}$, the Denjoy alternative holds outside a null set, i.e., for almost every real x, either the derivative of f exists at x, or the derivative fails to exist in the worst possible way: the limit superior of the slopes around x equals $+\infty$, and the limit inferior $-\infty$. Algorithmic randomness allows us to define randomness notions giving rise to different concepts of *almost everywhere*. It is then natural to wonder which of these concepts corresponds to the *almost everywhere* notion appearing in the Denjoy-Young-Saks theorem. To answer this question Demuth investigated effective versions of the theorem and proved that Demuth randomness is strong enough to ensure the Denjoy alternative for Markov computable functions. In this paper, we show that the set of these points is indeed strictly bigger than the set of Demuth random reals — showing that Demuth's sufficient condition was too strong — and moreover is incomparable with Martin-Löf randomness; meaning in particular that it does not correspond to any known set of random reals.

To prove these two theorems, we study density-type results, such as the Lebesgue density theorem and obtain results of independent interest. We show for example that the classical notion of Lebesgue density can be characterized by the only very recently defined notion of difference randomness. This is the first analytical characterization of difference randomness. We also consider the concept of porous points, a special type of Lebesgue non-density points for which drops in density are witnessed by single intervals. An essential part of our proof will be to argue that porous points of effectively closed classes can never be difference random.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Differentiability, Denjoy alternative, density, porosity, randomness

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.543

1 Introduction

The aim of the theory of algorithmic randomness is to give a precise definition of what it means for a single object (usually a finite or infinite binary sequence) to be random. For

* The second author is supported by a Feodor Lynen postdoctoral research fellowship by the Alexander von Humboldt Foundation.
† The third author is supported by the National Science Foundation under grant DMS-1001847.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 543–554

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

infinite binary sequences (or reals, as any real can be represented by an infinite binary sequence) a satisfactory definition was given by Martin-Löf [11]. Informally an infinite sequence x is Martin-Löf random if it does not belong to any set which can computably be shown to have measure 0. Even though Martin-Löf's definition is still believed to be the best one (at least the most well-behaved), many alternative notions of randomness have appeared in the literature over the years, some weaker than Martin-Löf randomness, some stronger. We refer the reader to the two recent books [8, 12] for an extensive survey of these notions.

An interesting line of research is to study the connections between algorithmic randomness and computable analysis. The latter is concerned with effective versions of classical theorems in analysis, i.e., analytical theorems where the objects involved (functions, sets, points, etc.) are effective, that is, computable in some sense. Consider a classical theorem of type "for any function f, for almost every x, ...". Its effective version will look like "for any effective function f, for almost every x, ...". Now, since there are only countably many effective functions (no matter what meaning is given to effective), one can reverse the quantifiers, and get a statement of type "for almost every x, for every effective function f, ...". Therefore, a sufficiently random x will satisfy the conclusion of the theorem. For each such theorem, we can thus look at the following question: *How much* randomness is needed for x to satisfy the conclusion of the theorem? A recent example is a result proven in [1] and [10] showing that Martin-Löf randomness is precisely the level of randomness needed to satisfy the most natural effective version of Birkhoff's ergodic theorem. Another is a result of Brattka, Miller and Nies [3], which is closely connected to this paper. They considered the effective version of the following theorem. If f is a non-decreasing function from $\mathbb{R}$ to $\mathbb{R}$, then f is differentiable almost everywhere. Following the above scheme, they studied the class of reals x such that every *computable* non-decreasing function f is differentiable at x, and were able to show that this class precisely coincides with the class of computably random reals. This was surprising as computable randomness had very few known characterizations other than its original definition, and in particular, no known analytical characterization.

Demuth [7] studied an effective version of a related theorem, the so-called Denjoy-Young-Saks theorem, which asserts that any function $f : \mathbb{R} \to \mathbb{R}$ satisfies the Denjoy alternative at almost all points. The Denjoy alternative at a point x states that either the function is differentiable at x or the derivative fails to exist in the most dramatic way, i.e., the function f has around x arbitrarily large positive slopes and negative slopes. Demuth mainly studied the Denjoy alternative for Markov computable functions (which we will define in a moment) and studied the set DA of points x such that any Markov computable function satisfies the Denjoy alternative at x. Demuth introduced a randomness notion, now called Demuth randomness, which he proved to be sufficient to be in DA. The main result of this paper is that Demuth randomness is in fact too strong a condition, and that the class DA is strictly larger than the class of Demuth random reals. Difference randomness is a notion of randomness slightly stronger than Martin-Löf randomness and significantly weaker than Demuth randomness. We show that this notion already implies the Denjoy alternative for Markov computable functions.

▶ **Theorem 1.** *Every difference random real belongs to* DA.

We then show that this result cannot be strengthened to Martin-Löf randomness: in fact, Martin-Löf randomness is neither sufficient nor necessary to ensure the Denjoy alternative for Markov computable functions.

▶ **Theorem 2.** *The set* DA *of reals that satisfy the Denjoy alternative for all Markov computable functions is incomparable under inclusion with the set of Martin-Löf random reals.*

These results will be proven in Section 3. Finally we show that x is difference random if and only if x is Martin-Löf random and has positive density in every effectively closed class in which x is contained.

1.1 Preliminaries

We provide notation, recall the definitions of computable and Markov computable functions on the real numbers, and recall the definitions of Martin-Löf randomness, difference randomness, and computable randomness.

Basic notation. The set of finite binary sequences (we also say strings) is denoted by $2^{<\omega}$, and the set of infinite binary sequences, called Cantor space, is denoted by 2^{ω}. For a string σ, $|\sigma|$ is the length of σ. If σ is a string and x is either a string or an infinite binary sequence, we say that σ is a prefix of x, which we write $\sigma \preceq x$, if the first $|\sigma|$ bits of x are exactly the string σ. Given a binary sequence, finite or infinite, with length at least n, $x{\restriction}n$ denotes the string made of the first n bits of x.

The Cantor space is classically endowed with the product topology. A basis of this topology is the set of cylinders: given a string $\sigma \in 2^{<\omega}$, the cylinder $[\![\sigma]\!]$ is the set of elements of 2^{ω} having σ as a prefix. If A is a set of strings, $[\![A]\!]$ is the union of the cylinders $[\![\sigma]\!]$ with $\sigma \in A$. The Lebesgue measure λ (or uniform measure) on the Cantor space is the probability measure assigning to each bit the value 0 with probability $1/2$ and the value 1 with probability $1/2$, independently of all other bits. Equivalently it is the measure λ such that $\lambda([\![\sigma]\!]) = 2^{-|\sigma|}$ for all σ. We abbreviate $\lambda([\![\sigma]\!])$ by $\lambda(\sigma)$. Given two subsets $\mathcal{X}$ and $\mathcal{Y}$, the second one being of positive measure, the conditional measure $\lambda(\mathcal{X}|\mathcal{Y})$ of $\mathcal{X}$ knowing $\mathcal{Y}$ is the quantity $\lambda(\mathcal{X} \cap \mathcal{Y})/\lambda(\mathcal{Y})$. As before, if $\mathcal{X}$ or $\mathcal{Y}$ is a cylinder $[\![\sigma]\!]$, we will simply write it as σ.

Computable real-valued functions. Most of the paper will focus on functions from $[0,1]$ to $\mathbb{R}$. The set $[0,1]$ is typically identified with 2^{ω}, where a real $x \in [0,1]$ is identified with its binary expansion. This extension is unique, except for dyadic rationals (of the form $a2^{-b}$ with a, b positive integers) which have two. A cylinder $[\![\sigma]\!]$ will be commonly identified with the open interval $(0.\sigma, 0.\sigma + 2^{-n})$, where $0.\sigma$ is the dyadic rational whose binary extension is $0.\sigma 000\ldots$.

We say that a function $f : [0,1] \to \mathbb{R}$ is computable (over the reals) if it can be effectively approximated with arbitrary precision. More precisely, f is computable (over the reals) if there exists a computable function $\widehat{f} : \mathbb{Q} \times \mathbb{N} \to \mathbb{Q}$ and a computable $\psi : \mathbb{N} \to \mathbb{N}$ such that for all $x \in [0,1]$ and $q \in [0,1] \cap \mathbb{Q}$, we have $|x - q| < 2^{-\psi(n)} \Rightarrow |f(x) - \widehat{f}(q,n)| < 2^{-n}$. Note that a computable function over the reals is by this definition necessarily continuous. A real x is computable if the constant function x is computable. Equivalently, a real is computable if its binary expansion, seen as a function from $\mathbb{N}$ to $\{0,1\}$ is computable.

We denote the set of computable reals by $\mathbb{R}_c$. The image of a computable real by a computable function is itself a computable real. Since the computable reals form a dense subset of the reals, a computable function is uniquely determined by its restriction $\mathbb{R}_c \to \mathbb{R}_c$. The class of Markov computable functions is a larger class of functions $\mathbb{R}_c \to \mathbb{R}_c$. As we just said, a real x is computable if there is a computable function β which computes its binary expansion. Any index i of β in an uniform enumeration $(\phi_i)_i$ of partial computable functions is called a *name* for x. A function $f : \mathbb{R}_c \to \mathbb{R}_c$ is said to be Markov computable if from a name of $x \in \mathbb{R}_c$, one can effectively compute a name for $f(x)$. More precisely, f is Markov computable if there exists a partial computable function $\varphi : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \mathbb{R}_c$, if i is a name for x, then $\varphi(i)$ is defined and is a name for $f(x)$. Given a

Markov computable function f, $x \in [0,1]$ and $s \in \mathbb{N}$, we sometimes use the notation $f(x)_s$ to denote the approximation of $f(x)$ at precision level 2^{-s}. Unless specified otherwise, a Markov computable function is always assumed to be total on $[0,1] \cap \mathbb{R}_c$. An important theorem of Tseitin [13] states that a total Markov computable function is always continuous on its domain.

We define the following analytical notations: for a function f, the *slope* at a pair a, b of distinct reals in its domain is

$$S_f(a,b) = \frac{f(a) - f(b)}{a - b}.$$

Recall the following definitions for the case that z is in the domain of f.

$$\overline{D}f(z) = \limsup_{h \to 0} S_f(z, z+h) \quad \text{and} \quad \underline{D}f(z) = \liminf_{h \to 0} S_f(z, z+h)$$

The derivative $f'(z)$ exists if and only if these values are equal and finite.

In this article we will work with functions that are not necessarily defined on all reals, e.g., Markov computable functions. When working with these functions $\overline{D}f(z)$ and $\underline{D}f(z)$ are not defined for all z. Nonetheless, in case the set $\mathrm{dom}(f)$ is a dense subset of $[0,1]$, one can consider the lower and upper *pseudo*-derivatives defined by:

$$\underset{\sim}{D}f(x) = \liminf_{h \to 0^+} \{S_f(a,b) \colon a, b \in \mathrm{dom}(f) \wedge\ a \leq x \leq b \wedge\ 0 < b - a \leq h\}.$$

$$\widetilde{D}f(x) = \limsup_{h \to 0^+} \{S_f(a,b) \colon a, b \in \mathrm{dom}(f) \wedge\ a \leq x \leq b \wedge\ 0 < b - a \leq h\}.$$

Note that when the function f is continuous on its (dense) domain, which is the case for computable and Markov computable functions, one can replace $\mathrm{dom}(f)$ by any dense subset of $\mathrm{dom}(f)$ in the definition of $\underset{\sim}{D}f$ and $\widetilde{D}f$. For Markov computable functions, for example, one could use $\mathbb{Q}$ instead of $\mathbb{R}_c$ to define the pseudo-derivatives.

As we have seen above, an open set $\mathcal{U} \subseteq 2^\omega$ is a union of cylinders. If it is a union of a *computably enumerable* (c.e.) family of cylinders, it is said to be effectively open (or c.e. open). A set is called effectively closed set if its complement is effectively open. We will need the following technical lemma. We omit the proof due to space considerations.

▶ **Lemma 3.** *Let $h \colon\ \subseteq [0,1] \to \mathbb{R}_0^+$ be a computable function that is defined and non-decreasing on an effectively closed class $\mathcal{C}$. Then $h \restriction_{\mathcal{C}}$ can be extended to a function $g \colon [0,1] \to \mathbb{R}_0^+$ that is computable and non-decreasing on $[0,1]$.*

Randomness notions. If $(\mathcal{U}_n)$ is a sequence of open sets, it is said to be a uniformly c.e. sequence of open sets if there is a sequence (W_n) of uniformly c.e. sets of strings such that each $\mathcal{U}_n$ is the union of the cylinders generated by the strings in W_n.

A *Martin-Löf test* is a uniformly c.e. sequence $(\mathcal{U}_n)_n$ of open sets such that for all n, $\lambda(\mathcal{U}_n) < 2^{-n}$. A *difference test* is a pair $((\mathcal{U}_n)_n, \mathcal{C})$ of a uniformly c.e. sequence $(\mathcal{U}_n)_n$ of open classes and a single effectively closed class $\mathcal{C}$ such that for all n, $\lambda(\mathcal{U}_n \cap \mathcal{C}) < 2^{-n}$. A strong test is a uniformly c.e. sequence $(\mathcal{U}_n)_n$ of open sets with the weaker condition that $\lim_n \lambda(\mathcal{U}_n) = 0$.

▶ **Definition 4.** A sequence $x \in 2^\omega$ is called *Martin-Löf random* if there is no Martin-Löf test *covering* it, i.e., for any Martin-Löf test $(\mathcal{U}_n)_n$ we have $x \notin \bigcap_n \mathcal{U}_n$. A sequence $x \in 2^\omega$ is called *weakly 2-random* if there is no strong test covering it, i.e., for any strong test $(\mathcal{U}_n)_n$ we have $x \notin \bigcap_n \mathcal{U}_n$. A sequence $x \in 2^\omega$ is called *difference random* if there is no difference test *covering* it, i.e., if for any difference test $((\mathcal{U}_n)_n, \mathcal{C})$ we have $x \notin \bigcap_n (\mathcal{U}_n \cap \mathcal{C})$.

The notion of difference randomness was introduced by Franklin and Ng [9]. They proved that the set of difference random reals in fact coincides with the set of Martin-Löf random reals that are Turing incomplete.

▶ **Proposition 5.** *For both Martin-Löf randomness and difference randomness, it is equivalent (see for example [8]) to require "almost avoidance": a sequence $x \in 2^\omega$ is Martin-Löf random (resp. difference random) if and only if for every Martin-Löf test $(\mathcal{U}_n)$ (resp. difference test $((\mathcal{U}_n), \mathcal{C})$), x only belongs to finitely many $\mathcal{U}_n$ (resp. finitely many $\mathcal{U}_n \cap \mathcal{C}$).*

Note that this type of "almost avoidance" variation of definitions is not admissible for weak 2-randomness.

Another strengthening of Martin-Löf randomness is Demuth randomness. A Demuth test is a sequence $(\mathcal{U}_n)$ of effectively open sets with $\lambda(\mathcal{U}_n) < 2^{-n}$ for all n, which is not necessarily uniformly c.e., but instead enjoys the following weak form of uniformity: there exists an ω-c.e. function $f : \mathbb{N} \to \mathbb{N}$ which for each n gives a c.e. index for a set of strings generating $\mathcal{U}_n$.

▶ **Definition 6.** A sequence $x \in 2^\omega$ is said to be Demuth random if for every Demuth test $(\mathcal{U}_n)$, x belongs to only finitely many $\mathcal{U}_n$.

The last notion of randomness we will discuss in the paper is computable randomness. Its definition involves the notion of martingale.

▶ **Definition 7.** A martingale is a function $d : 2^{<\omega} \to [0, \infty)$ such that for all $\sigma \in 2^{<\omega}$

$$d(\sigma) = \frac{d(\sigma 0) + d(\sigma 1)}{2}.$$

Intuitively, a martingale represents a betting strategy where a player successively bets money on the values of the bits of an infinite binary sequence (doubling its stake when the guess is correct); $d(\sigma)$ then represents the capital of the player after betting on initial segment σ. With this intuition, a martingale succeeds against a sequence x if $\limsup_n d(x\upharpoonright n) = +\infty$. A computably random sequence is a sequence against which no computable betting strategy succeeds. In other words:

▶ **Definition 8.** A sequence $x \in 2^\omega$ is computably random if and only if for every computable martingale d, $\limsup_n d(x\upharpoonright n) < +\infty$.

We denote by MLR, W2R, DiffR, DemR, CR the classes of Martin-Löf random, weakly 2-random, difference random, Demuth random and computably random sequences respectively.

Given a sequence $x \in 2^\omega$, the following implications

$$\begin{array}{ccccccc} x \in \mathsf{W2R} & & & & & & \\ & \searrow & & & & & \\ & & x \in \mathsf{DiffR} & \longrightarrow & x \in \mathsf{MLR} & \longrightarrow & x \in \mathsf{CR} \\ & \nearrow & & & & & \\ x \in \mathsf{DemR} & & & & & & \end{array}$$

hold and no other implication holds in general (other than those which can be derived by transitivity from the above diagram). See for example [12] for a detailed exposition.

2 Density and porosity

In this section, we initiate the study of effective aspects of Lebesgue density, which will be crucial in the proofs of Theorems 1 and 2. In this section, we mostly focus on what is needed for the proofs of these theorems. In Section 4 we will provide further results on density.

Let us first recall the concept of Lebesgue density.

▶ **Definition 9.** We define the (lower Lebesgue) density ρ of a set $\mathcal{C} \subseteq \mathbb{R}$ at a point x to be the quantity

$$\rho(x|\mathcal{C}) := \liminf_{\gamma,\delta\to 0^+} \frac{\lambda([x-\gamma, x+\delta] \cap \mathcal{C})}{\lambda([x-\gamma, x+\delta])},$$

where λ is the Lebesgue measure.

Intuitively, this measures what fraction of the space is filled by $\mathcal{C}$ around x if we "zoom in" arbitrarily close. Note that the density of a set at a point is between 0 and 1.

Again, in the rest of the paper, we will freely identify 2^ω and $[0,1]$ and will therefore be able to talk about density of a set $\mathcal{C} \subseteq 2^\omega$ at a point.

▶ **Theorem 10** (Lebesgue density theorem)**.** *Let $\mathcal{C} \subseteq \mathbb{R}$ be a measurable set. Then $\rho(x|\mathcal{C}) = 1$ for all points $x \in \mathcal{C}$ outside a set of measure 0.*

The concept of porosity of a set at a point forms a cornerstone of the proofs of Theorems 1 and 2. The following definition originates in the work of Denjoy. See for instance [2, 5.8.124] (but note the typo in the definition there); also [4, Ex. 7:9.12].

▶ **Definition 11.** We say that $\mathcal{C}$ is *porous at x* via $\varepsilon > 0$ if for each $\alpha > 0$ there exists β with $0 < \beta \leq \alpha$ such that $(x-\beta, x+\beta)$ contains an open interval of length $\varepsilon\beta$ that is disjoint from $\mathcal{C}$. We say that $\mathcal{C}$ is porous at x if it is porous at x via some ε. We call a *non-porosity point* a real x such that every effectively closed class to which it belongs is non-porous at x.

Clearly porosity of $\mathcal{C}$ at x implies $\rho(x|\mathcal{C}) < 1$. Therefore for almost every point x of a measurable class $\mathcal{C}$, we have that $\mathcal{C}$ is not porous at x.

▶ **Lemma 12.** *Let $\mathcal{C} \subseteq [0,1]$ be an effectively closed class. If $z \in \mathcal{C}$ is difference random, then $\mathcal{C}$ is not porous at z.*

Proof. In this proof, we say that a string σ meets $\mathcal{C}$ if $[\![\sigma]\!] \cap \mathcal{C} \neq \emptyset$.

Fix $c \in \mathbb{N}$ such that $\mathcal{C}$ is porous at z via 2^{-c+2}. For each string σ consider the set of minimal "porous" extensions at stage t,

$$N_t(\sigma) = \left\{ \rho \succeq \sigma \;\middle|\; \exists \tau \succeq \sigma \left[\begin{array}{c} |\tau| = |\rho| \wedge |0.\tau - 0.\rho| \leq 2^{-|\tau|+c} \wedge \\ [\![\tau]\!] \cap \mathcal{C}_t = \emptyset \wedge \rho \text{ is minimal with this property} \end{array} \right] \right\}.$$

We claim that

$$\sum_{\substack{\rho \in N_t(\sigma) \\ \rho \text{ meets } \mathcal{C}}} 2^{-|\rho|} \leq (1 - 2^{-c-2}) 2^{-|\sigma|}. \tag{1}$$

To see this, let R be the set of strings ρ in (1). Let V be the set of prefix-minimal strings that occur as witnesses τ in (1). Then the open sets generated by R and by V are disjoint. Thus, if r and v denote their measures, respectively, we have $r + v \leq 2^{-|\sigma|}$. By definition of $N_t(\sigma)$, for each $\rho \in R$ there is $\tau \in V$ such that $|0.\tau - 0.\rho| \leq 2^{-|\tau|+c}$. This implies $r \leq 2^{c+1} v$. The two inequalities together imply (1).

Note that by the formal details of this definition even the "holes" τ are ρ's, and therefore contained in the sets $N_t(\sigma)$. This will be essential for the proof of the first of the following two claims. At each stage t of the construction we define recursively a sequence of anti-chains as follows.

$$B_{0,t} = \{\emptyset\}, \text{ and for } n > 0\colon B_{n,t} = \bigcup \{N_t(\sigma)\colon \sigma \in B_{n-1,t}\}$$

Claim. If a string ρ is in $B_{n,t}$ then it has a prefix ρ' in $B_{n,t+1}$.
This is clear for $n = 0$. Suppose inductively that it holds for $n-1$. Suppose further that ρ is in $B_{n,t}$ via a string $\sigma \in B_{n-1,t}$. By the inductive hypothesis there is $\sigma' \in B_{n-1,t+1}$ such that $\sigma' \preceq \sigma$. Since $\rho \in N_t(\sigma)$, ρ is a viable extension of σ' at stage $t+1$ in the definition of $N_{t+1}(\sigma')$, except maybe for the minimality. Thus there is $\rho' \preceq \rho$ in $N_{t+1}(\sigma')$. ◊

Claim. For each n, t, we have $\sum\{2^{-|\rho|} : \rho \in B_{n,t} \wedge \rho \text{ meets } \mathcal{C}\} \leq (1 - 2^{-c-2})^n$.
This is again clear for $n = 0$. Suppose inductively it holds for $n-1$. Then, by (1),

$$\sum_{\substack{\rho \in B_{n,t} \\ \rho \text{ meets } \mathcal{C}}} 2^{-|\rho|} = \sum_{\substack{\sigma \in B_{n-1,t} \\ \sigma \text{ meets } \mathcal{C}}} \sum_{\substack{\rho \in N_t(\sigma) \\ \rho \text{ meets } \mathcal{C}}} 2^{-|\rho|} \leq \sum_{\substack{\sigma \in B_{n-1,t} \\ \sigma \text{ meets } \mathcal{C}}} 2^{-|\sigma|}(1 - 2^{-c-2}) \leq (1 - 2^{-c-2})^n.$$

This establishes the claim. ◊

Now let $U_n = \bigcup_t [\![B_{n,t}]\!]$. Clearly the sequence $(U_n)_{n\in\mathbb{N}}$ is uniformly effectively open. By the first claim, $U_n = \bigcup_t [\![B_{n,t}]\!]$ is a nested union, so the second claim implies that $\lambda(\mathcal{C} \cap U_n) \leq (1 - 2^{-c-2})^n$. We show $z \in \bigcap U_n$ by induction on n. Clearly $z \in U_0$. If $n > 0$ suppose inductively $\sigma \prec z$ where $\sigma \in \bigcup_t B_{n-1,t}$. Since z is random there is η such that $\sigma \prec \eta \prec z$ and η ends in 0^c1^c. Every interval $(a, b) \subseteq [0, 1]$ contains an interval of the form $[\![\rho]\!]$ for a dyadic string ρ such that the length of $[\![\rho]\!]$ is no less than $(b - a)/4$. Thus, since $\mathcal{C}$ is porous at z via 2^{-c+2}, there is t, $\rho \succeq \eta$ and τ satisfying the condition in the definition of $N_t(\sigma)$. By the choice of η one verifies that $\tau \succeq \sigma$. Thus $z \in U_n$.

Now take a computable subsequence $(U_{g(n)})_{n\in\mathbb{N}}$ such that $\lambda(\mathcal{C} \cap U_{g(n)}) \leq 2^{-n}$ to obtain a difference test that z fails. ◀

3 Effective forms of the Denjoy-Young-Saks Theorem

We begin with the formal definition of the Denjoy alternative.

▶ **Definition 13.** Let $f :\subseteq [0, 1] \to \mathbb{R}$ be a partial function whose domain is dense. We say that f satisfies the Denjoy alternative at x if

- either the pseudo-derivative of f at x exists (meaning that $\widetilde{D}f(x) = \underset{\sim}{D}f(x)$)
- or $\widetilde{D}f(x) = +\infty$ and $\underset{\sim}{D}f(x) = -\infty$.

Intuitively this means that there are two ways for the alternative to hold: either the function behaves well on x by having a derivative at this place, or, if it behaves badly, it does so in the worst possible way, that being the fact that the limit superior and the limit inferior diverge as much as possible. The Denjoy-Young-Saks theorem (see, e.g., Bruckner [5]) states that the Denjoy alternative holds at almost all points for *any* function f.

3.1 Computable randomness means that all computable functions satisfy the Denjoy alternative

▶ **Definition 14** (Demuth [6]). A real $z \in [0, 1]$ is called Denjoy random (or a Denjoy set) if for no Markov computable function g we have $\underset{\sim}{D}g(z) = +\infty$.

In a preprint by Demuth [6, p. 6] it is shown that if $z \in [0, 1]$ is Denjoy random, then for every computable $f : [0, 1] \to \mathbb{R}$ the Denjoy alternative holds at z. By combining this result with the results in [3] we can achieve the following result that provides a pleasing characterization of computable randomness through differentiability of computable functions.

▶ **Theorem 15** (Demuth, Miller, Nies, Kučera)**.** *The following are equivalent for a real* $z \in [0,1]$.

1. *z is Denjoy random.*
2. *z is computably random.*
3. *For every computable $f\colon [0,1] \to \mathbb{R}$ the Denjoy alternative holds at z.*

Note that all we need for (2) $\Rightarrow$ (1) is that $f(q)$ is a computable real uniformly in a rational $q \in [0,1] \cap \mathbb{Q}$. Thus, in Definition 14 we can replace Markov computability of f by this weaker hypothesis.

3.2 Difference randomness implies that all Markov computable functions satisfy the Denjoy alternative

We now turn to the proof of Theorem 1. It will be enough to prove the following.

▶ **Proposition 16.** *Let $x \in 2^\omega$ be a computably random real that is also a non-porosity point. Then $x \in \mathsf{DA}$, i.e., all Markov computable functions satisfy the Denjoy alternative at x.*

To get Theorem 1 from this proposition, remember that a difference random point is always computably random, and, by Lemma 12 a difference random real is also always a non-porosity point.

Proof. We first prove a lemma, which takes advantage of the special way in which a set is arranged around its non-porosity points.

▶ **Lemma 17.** *Suppose that $f\colon \subseteq [0,1] \to \mathbb{R}$ is Markov computable. Let $\mathcal{C} \subseteq [0,1]$ be an effectively closed class such that there is an n with $\underset{\sim}{D}f(z) > -n$ for all $z \in \mathcal{C}$. Let the computably random real $x \in \mathcal{C}$ be a non-porosity point of $\mathcal{C}$. Then f is differentiable at x.*

Proof. We effectivize an argument of Bogachev [2, p. 371]. Replacing f by $f(x) + (n+1)x$, we may assume that for $x \in \mathcal{C}$ and some r and s, we have

$$\forall a, b\ [r < a \leq x \leq b < s \to S_f(a,b)_0 > 0].$$

We restrict our attention to the interval $[r,s]$; for notational simplicity we assume that $[r,s] = [0,1]$. Let $f_*(x) = \sup_{a \leq x} f(a)$. Then f_* is nondecreasing on $\mathcal{C}$.

Claim. The function $f_* \restriction_{\mathcal{C}}$ is computable.
To see this, recall that p, q range over $[0,1] \cap \mathbb{Q}$, and let $f^*(x) = \inf_{q \geq x} f(q)$. If $x \in \mathcal{C}$ and $f_*(x) < f^*(x)$ then x is computable: fix a rational d in between these two values. Then for $p, q \in (r,s)$ we have $p < x \leftrightarrow f(p) < d$, and $q > x \leftrightarrow f(q) > d$. Hence x is both left-c.e. and right-c.e., and therefore computable. Now a Markov computable function is continuous at every computable x. Thus $f_*(x) = f^*(x)$ for each x in $\mathcal{C}$.

To compute $f_*(x)$ for $x \in \mathcal{C}$ up to precision 2^{-n}, we can now simply search for rationals $p < x < q$ such that $0 < f(q)_{n+2} - f(p)_{n+2} < 2^{-n-1}$, and output $f(p)_{n+2}$. If during this search we detect that $x \notin \mathcal{C}$, we stop. This shows the claim. ◊

By Lemma 3 there is a computable nondecreasing function g defined on $[0,1]$ that extends f_*. Then by a classic theorem of Lebesgue, $g'(x)$ exists for a.e. $x \in [0,1]$. A result by Brattka, Miller and Nies [3, Thm. 4.1] states that in fact computable randomness of x is enough to guarantee the existence of $g'(x)$.

It therefore suffices to show that for each $x \in \mathcal{C}$ such that $g'(x)$ is defined and $\mathcal{C}$ is not porous at x, we have $\widetilde{D}f(x) \leq g'(x) \leq \underset{\sim}{D}f(x)$. Since $\underset{\sim}{D}f(x) \leq \widetilde{D}f(x)$, this would establish the theorem.

To see this, we show $\widetilde{D}f(x) \leq g'(x)$, the other inequality being symmetric. Fix $\varepsilon > 0$. Choose $\alpha > 0$ such that

$$\forall u, v \in \mathcal{C}\,[(u \leq x \leq v \wedge 0 < v - u \leq \alpha) \to S_{f_*}(u, v) \leq g'(x)(1 + \varepsilon)] \tag{2}$$

furthermore, since $\mathcal{C}$ is not porous at x, we may assume that for each $\beta \leq \alpha$, the interval $(x - \beta, x + \beta)$ contains no open subinterval of length $\varepsilon\beta$ that is disjoint from $\mathcal{C}$. Now suppose that $a, b \in [0, 1] \cap \mathbb{Q}$, $a < x < b$ and $\beta = 2(b - a) \leq \alpha$. There are $u, v \in \mathcal{C}$ such that $0 \leq a - u \leq \varepsilon\beta$ and $0 \leq v - b \leq \varepsilon\beta$. Since $u, v \in \mathcal{C}$ we have $f_*(u) \leq f(a)$ and $f(b) \leq f_*(v)$. Therefore $v - u \leq b - a + 2\varepsilon\beta = (b - a)(1 + 4\varepsilon)$. It follows that

$$S_f(a, b) \leq \frac{f_*(v) - f_*(u)}{b - a} \leq S_{f_*}(u, v)(1 + 4\varepsilon) \leq g'(x)(1 + 4\varepsilon)(1 + \varepsilon). \qquad \blacktriangleleft$$

We are now ready to prove Proposition 16. Suppose x is computably random and is a non-porosity point. Let f be a Markov computable function. Suppose that f does not satisfy the Denjoy alternative at x by strong failure of the existence of the pseudo-derivative at x. We therefore are $\underset{\sim}{D}f(x) > -\infty$ or $\widetilde{D}f(x) < +\infty$. Suppose the first one holds (the proof for the other case is similar), and take an n such that $\underset{\sim}{D}f(x) > -n$. By definition of $\underset{\sim}{D}$, this means that for some fixed positive rational ε and some fixed t, x belongs to the effectively closed class:

$$\mathcal{C} = \{x \in [0, 1] \mid \forall q_1, q_2 \in \mathbb{Q}\ s.t.\ x \in [q_1, q_2] \wedge |q_2 - q_1| < \varepsilon,\ S_f(q_1, q_2)_t \geq -n\}$$

We can therefore apply Proposition 16 to this class $\mathcal{C}$ (every point $z \in \mathcal{C}$ is such that $\underset{\sim}{D}f(z) > -n$, x belongs to $\mathcal{C}$, x is computably random and is a non-porosity point). Therefore, f is differentiable at x, and thus the Denjoy alternative holds indeed. $\blacktriangleleft$

3.3 The class DA is incomparable with the Martin-Löf random reals

▶ **Theorem 18.** *There exists a real x that is not Martin-Löf random but nonetheless satisfies the Denjoy alternative for all Markov computable functions.*

Proof sketch. The Denjoy alternative at x can be met in two ways. We will say "the DA for f is fulfilled by existence" if the (pseudo-)derivative of f at x exists and say that "the DA for f is fulfilled by failure" in the other case. To prove the statement we construct a set x by forcing that is CR, not MLR and for every Markov computable function either fulfills the DA by failure for this function or is a density point (and therefore certainly not a porosity point) of a certain effectively closed class L such that L and f fulfill the requirements of Lemma 17. The argument to prove the statement then goes like this: if we fulfill the DA by failure we are done. Otherwise x would be a density point of L. Since $x \in$ CR we can invoke Lemma 17 to show that f is differentiable and therefore fulfills the DA by existence.

Assume we have constructed the initial segment σ of x so far, and are given a computable martingale M. The most interesting part of the argument is how to ensure that we are density points of certain effectively closed classes of the form $L := \{x \succeq \sigma \mid M(x\upharpoonright n) < \varepsilon \text{ for all } n > |\sigma|\}$. To do this we need to make sure that the density of x in L will be 1 in the limit. At every stage of the construction we will make sure that the density of x is at least $1 - q$ for some q by choosing the right extension σ' of σ. When we later extend σ' further we will make q smaller and smaller while forever staying inside L. This way in the limit we reach density 1 in L.

To achieve density $1 - q$ as required, we look at the quantity $m := \inf_{\tau \succeq \sigma} M(\tau)$. We interpret m as an amount of capital that the martingale M has put on a savings account

and is not touching anymore. Of course this implies that M also has less capital available for betting and can therefore reach capital ε only on a smaller fraction of the extensions of σ. By applying the Ville-Kolmogorov inequality for martingales to $M - m$ it is clear that M can reach capital ε only on a set of extensions of σ of relative measure $1 - \frac{M(\sigma)-m}{\varepsilon-m}$. Or, in other words, σ has density $1 - (M(\sigma) - m)/(\varepsilon - m)$ in L. By replacing σ with a long enough extension we can make sure that $M(\sigma)$ is arbitrarily close to m and thereby raise the density to the desired level $1 - q$. Although this method only controls the density in L of the dyadic intervals containing x, we are able to show that this is sufficient. ◀

▶ **Theorem 19.** *There exists a Markov computable function f for which the Denjoy alternative does not hold at Chaitin's Ω. Moreover, f can be taken to be uniformly continuous, i.e., it can be built in such a way that it has a (unique) continuous extension to $[0, 1]$.*

Proof. Let $(\mathcal{U}_n)$ be a universal Martin-Löf test, i.e., a test such that all reals that are not in MLR are covered by it (the existence of such a Martin-Löf test is well-known). No computable real can be Martin-Löf random; every $x \in \mathbb{R}_c$ belongs to $\mathcal{U}_1$. Let Ω be the leftmost point of the complement of $\mathcal{U}_1$. It is not hard to see that since $\mathcal{U}_1$ is a c.e. open set, it is an effective union $\bigcup_t I_t$ of *closed* rational intervals I_t that overlap only on their endpoints. Let $\mathcal{U}_1[s] = \bigcup_{t<s} I_t$ and let Ω_s be the leftmost point of $\mathcal{U}_1[s]$. Then Ω is approximated from below by the computable sequence of rationals $(\Omega_s)_s$.

Our function f is defined as the restriction to $\mathbb{R}_c$ of the following function F. Outside $\mathcal{U}_1$, F is equal to 0. On $\mathcal{U}_1$, it is constructed sequentially as follows. At stage $s + 1$, consider I_s. There are two cases.

1. Either adding this interval does not change the value of Ω (i.e., $\Omega_{s+1} = \Omega_s$). In that case, define the function F to be equal to zero on I_s.
2. Or, this interval does change the value of Ω: $\Omega_{s+1} > \Omega_s$. In this case, define F on I_s to be the triangular function taking value 0 on the endpoints of I_s and reaching the value v at the middle point, where v is defined as follows. Let t be the last stage at which the *previous* increase of Ω occurred (i.e., t is maximal such that $t < s$ and $\Omega_{t+1} > \Omega_t$). Let n be the smallest integer such that the real interval $[\Omega_t, \Omega_{t+1}]$ contains a multiple of 2^{-n}. For that n, set $v = 2^{-n/2}$.

First, we see that the restriction f of F to $\mathbb{R}_c$ is Markov computable: given a code i, we try to compute the real x coded by i (remember that such an x might not exist) until we find a sufficiently good estimate $a < x < b$ such that the interval $[a, b]$ is contained either in one or in the union of two of the intervals appearing in the enumeration of $\mathcal{U}_1$. It is then easy to compute F at x as one can decide which of the above cases hold for each interval, and both the zero function and the triangular function are computable on $\mathbb{R}_c$. (In the triangular case, note that the value n of the construction can be found computably.)

We claim that the function f does not satisfy the Denjoy alternative at Ω. More precisely, we have $\widetilde{D}f(\Omega) = 0$ and $\underset{\sim}{D}f(\Omega) = -\infty$. Notice that f is equal to 0 on $(\Omega, 1] \cap \mathbb{R}_c$ and non-negative on $[0, \Omega) \cap \mathbb{R}_c$, taking the value 0 at computably real points arbitrarily close to Ω (at least the endpoints of intervals I_s enumerated on the left of Ω), therefore $\widetilde{D}f(\Omega) = 0$ is clear. To see that $\underset{\sim}{D}f(\Omega) = -\infty$, consider for all k the dyadic real a_k which is a multiple of 2^{-k}, is smaller than Ω and such that $\Omega - a_k < 2^{-k}$. Since $a_k < \Omega$, there exists a stage t such that $a_n \in [\Omega_t, \Omega_{t+1}]$. Let $s > t$ be the next stage at which Ω increases. By definition, F is then defined to be a triangular function on $[\Omega_s, \Omega_{s+1}]$ of height $2^{-n/2}$.Thus, letting x_k be middle point of $[\Omega_s, \Omega_{s+1}]$ and $q > \Omega$ be a rational such that $q - a_k < 2^{-k}$, we have

$$S_f(x_k, q) = \frac{f(q) - f(x_k)}{q - x} \leq \frac{0 - 2^{-k/2}}{2^{-k}} \leq -2^{k/2}.$$

Since this happens for all k, we have $\underset{\sim}{D}f(\Omega) = -\infty$.

It remains to show that the function F is continuous on $[0,1]$. But this is almost immediate as one can write $F = \sum_n h_n$ where h_n is the function equal to 0 except on the intervals on which F is a triangular function of height $2^{-n/2}$, and on that interval $h_n = F$. It is obvious that the h_n are continuous and of magnitude at most $2^{-n/2}$. Therefore $\sum_n ||h_n|| < \infty$, so by the Weierstrass M-test we can conclude that the convergence is uniform and hence the function $\sum_n h_n$ is continuous. ◀

4 Positive density as a randomness notion

We return to the notion of positive density and give an interesting characterization of incomplete Martin-Löf random sets.

▶ **Theorem 20.** *The following statements are equivalent for a Martin-Löf random real x.*

1. *x is difference random.*
2. *x has incomplete Turing degree.*
3. *x is a point of positive lower Lebesgue density in every effectively closed class $\mathcal{C}$ with $x \in \mathcal{C}$, that is, $\rho(x|\mathcal{C}) \neq 0$.*

Proof sketch. The equivalence between (1) and (2) has been shown by Franklin and Ng [9].

(1) ⇒ (3): Proof by contraposition. Assume that $x \in \mathsf{MLR}$ and that for all ε there is an interval I with $x \in I$ such that $\lambda(\mathcal{C} \mid I) < \varepsilon$. Then for all k let

$$U_k = \{z \mid \exists \text{ interval } I \colon z \in I \text{ and } \lambda(\mathcal{C} \mid I) < 2^{-k}\}.$$

Since $\mathcal{C}$ is effectively closed, these classes are uniformly effectively open; and clearly $x \in \mathcal{C}$. The measure bound on this difference test follows from the following lemma, the proof of which we omit due to space considerations.

▶ **Lemma 21.** *Let $\mathcal{C} \subseteq [0,1]$ be closed. Let $U_k = \{z \mid \exists \text{ interval } I \colon z \in I \text{ and } \lambda(\mathcal{C} \mid I) < 2^{-k}\}$. Then $\lambda(\mathcal{C} \cap U_k) \leq 2^{-k+1}$.*

(3) ⇒ (2): We only sketch the proof due to space considerations.

Suppose now that x is Martin-Löf random and Turing complete. We are going to show that x has lower density 0 inside some effectively closed class $\mathcal{C}$. We show that, given a rational ε, we can effectively construct an effectively closed class $\mathcal{C}_\varepsilon$ such that $x \in \mathcal{C}_\varepsilon$ and $\lambda(\mathcal{C}_\varepsilon \mid x\upharpoonright n) < \varepsilon$ for some n. It will then suffice to let $\mathcal{C} := \bigcap_\varepsilon \mathcal{C}_\varepsilon$ for an effective list of ε's that converge to 0.

Fix $\varepsilon > 0$. In this construction, we will build an auxiliary c.e. set W. By the recursion theorem, since x is complete, we can assume to know in advance a Turing reduction Γ such that $\Gamma^x = W$.

In order to lower the density of $\mathcal{C}_\varepsilon$ around x we need to remove many reals from $\mathcal{C}_\varepsilon$. Since we do not know x, this comes at the risk of inadvertently removing x as well. The approach of the proof is then to make use of the fact that we control W. We keep observing the results of the reduction Γ relative to all possible oracles in a neighborhood and wait until we see a certain type of behavior (the reduction outputs 0 on a certain value) on all oracles except fraction ε. As soon as this happens we change W in such a way that it does exactly *not* show this behavior. Since x computes W it certainly cannot be among the $1-\varepsilon$ fraction of oracles showing the special behavior, so we can safely remove them from $\mathcal{C}_\varepsilon$.

Of course it must be avoided that we wait forever, since in that case the measure of $\mathcal{C}_\varepsilon$ would forever remain equal to 1. It will therefore be necessary to argue why we can be sure

that we will eventually observe the special behavior. To see this, we will argue that if we never observe that behavior, x is in a descending chain of sets U_k such that U_k always has measure $1-\varepsilon$ relative to U_{k-1}, and that this chain actually is a Martin-Löf test covering x. This of course contradicts $x \in \mathsf{MLR}$. ◀

Together with Lemma 12 we get the following corollary. To the best of our knowledge there exists no direct proof of this fact.

▶ **Corollary 22.** *For any $x \in \mathsf{MLR}$ the following implication holds: If for all effectively closed classes $\mathcal{C}$ with $x \in \mathcal{C}$ it holds that $\rho(x|\mathcal{C}) > 0$ then for all effectively closed classes $\mathcal{C}$ with $x \in \mathcal{C}$ we have that $\mathcal{C}$ is not porous at x.*

▶ Remark. If x is weakly 2-random and $\mathcal{C}$ is an effectively closed class containing x, then $\rho(x|\mathcal{C}) = 1$. This is because for x to have $\rho(x|\mathcal{C}) < 1$ in some $\mathcal{C}$ can be written as $\forall \delta_0\ \exists \delta < \frac{\delta_0}{2} : \frac{\lambda([x-\delta,x+\delta]\cap\mathcal{C})}{\lambda([x-\delta,x+\delta])} < 1-\varepsilon$ for some fixed ε, which is a Π^0_2 condition. By the Lebesgue density theorem, the set of these x (for each $\mathcal{C}$) is also null, so they are covered by a strong test.

References

1 Laurent Bienvenu, Adam Day, Mathieu Hoyrup, Ilya Mezhirov, and Alexander Shen. Ergodic-type characterizations of algorithmic randomness. To appear in Information and Computation.

2 Vladimir I. Bogachev. *Measure theory. Vol. I, II.* Springer-Verlag, Berlin, 2007.

3 Vasco Brattka, Joseph S. Miller, and André Nies. Randomness and differentiability. Submitted.

4 Andrew Bruckner, Judith Bruckner, and Brian Thomson. *Real Analysis.* Prentice Hall (Pearson), 2007.

5 Andrew M. Bruckner. *Differentiation of real functions*, volume 659 of *Lecture Notes in Mathematics*. Springer, Berlin, 1978.

6 Osvald Demuth. *Preprint for* Remarks on Denjoy sets. Technical report, 1988.

7 Osvald Demuth. Reducibilities of sets based on constructive functions of a real variable. *Commentationes Mathematicae Universitatis Carolinae*, 29(1):143–156, 1988.

8 Rodney Downey and Denis Hirschfeldt. *Algorithmic randomness and complexity.* Theory and Applications of Computability. Springer, 2010.

9 Johanna Franklin and Keng Meng Ng. Difference randomness. *Proceedings of the AMS*, 139:345–360, 2011.

10 Johanna N.Y. Franklin, Noam Greenberg, Joseph S. Miller, and Keng Meng Ng. Martin-Löf random points satisfy Birkhoff's ergodic theorem for effectively closed sets. To appear in Proceedings of the American Mathematical Society.

11 Per Martin-Löf. The definition of random sequences. *Information and Control*, 9:602–619.

12 André Nies. *Computability and randomness.* Oxford Logic Guides. Oxford University Press, 2009.

13 G. S. Tseitin. Algorithmic operators in constructive metric spaces. *Transactions of the American Mathematical Society*, 64, 1967.

The Determinacy of Context-Free Games

Olivier Finkel [1]

1 **Equipe de Logique Mathématique**
Institut de Mathématiques de Jussieu
CNRS et Université Paris 7, France.
finkel@logique.jussieu.fr

Abstract

We prove that the determinacy of Gale-Stewart games whose winning sets are accepted by real-time 1-counter Büchi automata is equivalent to the determinacy of (effective) analytic Gale-Stewart games which is known to be a large cardinal assumption. We show also that the determinacy of Wadge games between two players in charge of ω-languages accepted by 1-counter Büchi automata is equivalent to the (effective) analytic Wadge determinacy. Using some results of set theory we prove that one can effectively construct a 1-counter Büchi automaton $\mathcal{A}$ and a Büchi automaton $\mathcal{B}$ such that: (1) There exists a model of ZFC in which Player 2 has a winning strategy in the Wadge game $W(L(\mathcal{A}), L(\mathcal{B}))$; (2) There exists a model of ZFC in which the Wadge game $W(L(\mathcal{A}), L(\mathcal{B}))$ is not determined. Moreover these are the only two possibilities, i.e. there are no models of ZFC in which Player 1 has a winning strategy in the Wadge game $W(L(\mathcal{A}), L(\mathcal{B}))$.

1998 ACM Subject Classification F.1.1 Models of Computation; F.4.1 Mathematical Logic

Keywords and phrases Automata and formal languages, logic in computer science, Gale-Stewart games, Wadge games, determinacy, context-free games

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.555

1 Introduction

Two-players infinite games have been much studied in Set Theory and in Descriptive Set Theory, see [9, 8]. In particular, if X is a (countable) alphabet having at least two letters and $A \subseteq X^\omega$, then the Gale-Stewart game $G(A)$ is an infinite game with perfect information between two players. Player 1 first writes a letter $a_1 \in X$, then Player 2 writes a letter $b_1 \in X$, then Player 1 writes $a_2 \in X$, and so on ... After ω steps, the two players have composed an infinite word $x = a_1 b_1 a_2 b_2 \ldots$ of X^ω. Player 1 wins the play iff $x \in A$, otherwise Player 2 wins the play. The game $G(A)$ is said to be determined iff one of the two players has a winning strategy. A fundamental result of Descriptive Set Theory is Martin's Theorem which states that every Gale-Stewart game $G(A)$, where A is a Borel set, is determined [9].

On the other hand, in Computer Science, the conditions of a Gale Stewart game may be seen as a specification of a reactive system, where the two players are respectively a non terminating reactive program and the "environment". Then the problem of the synthesis of winning strategies is of great practical interest for the problem of program synthesis in reactive systems. In particular, if $A \subseteq X^\omega$, where X is here a finite alphabet, and A is effectively presented, i.e. accepted by a given finite machine or defined by a given logical formula, the following questions naturally arise, see [15, 10]: (1) Is the game $G(A)$ determined? (2) If Player 1 has a winning strategy, is it effective, i.e. computable? (3) What are the amounts of space and time necessary to compute such a winning strategy? Büchi and Landweber gave a solution to the famous Church's Problem, posed in 1957, by stating that in a Gale Stewart game $G(A)$, where A is a regular ω-language, one can decide who the winner is and compute

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 555–566

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


a winning strategy given by a finite state transducer, see [16] for more information on this subject. In [15, 10] Thomas and Lescow asked for an extension of this result where A is no longer regular but deterministic context-free, i.e. accepted by some deterministic pushdown automaton. Walukiewicz extended Büchi and Landweber's Theorem to this case by showing first in [18] that that one can effectively construct winning strategies in parity games played on pushdown graphs and that these strategies can be computed by pushdown transducers. Notice that later some extensions to the case of higher-order pushdown automata have been established [1].

In this paper, we first address the question (1) of the determinacy of Gale-Stewart games $G(A)$, where A is a context-free ω-language accepted by a (non-deterministic) pushdown automaton, or even by a 1-counter automaton. Notice that there are some context-free ω-languages which are (effective) analytic but non-Borel and thus the determinacy of these games cannot be deduced from Martin's Theorem of Borel determinacy. On the other hand, Martin's Theorem is provable in ZFC, the commonly accepted axiomatic framework for Set Theory in which all usual mathematics can be developped. But the determinacy of Gale-Stewart games $G(A)$, where A is an (effective) analytic set, is not provable in ZFC; Martin and Harrington have proved that it is a large cardinal assumption equivalent to the existence of a particular real, called the real $0^\sharp$, see [8, page 637]. We prove here that the determinacy of Gale-Stewart games $G(A)$, whose winning sets A are accepted by real-time 1-counter Büchi automata, is equivalent to the determinacy of (effective) analytic Gale-Stewart games and thus also equivalent to the existence of the real $0^\sharp$.

Next we consider Wadge games which were firstly studied by Wadge in [17] where he determined a great refinement of the Borel hierarchy defined via the notion of reduction by continuous functions. These games are closely related to the notion of reducibility by continuous functions. For $L \subseteq X^\omega$ and $L' \subseteq Y^\omega$, L is said to be Wadge reducible to L' iff there exists a continuous function $f : X^\omega \to Y^\omega$, such that $L = f^{-1}(L')$; this is then denoted by $L \leq_W L'$. On the other hand, the Wadge game $W(L, L')$ is an infinite game with perfect information between two players, Player 1 who is in charge of L and Player 2 who is in charge of L'. And it turned out that Player 2 has a winning strategy in the Wadge game $W(L, L')$ iff $L \leq_W L'$. It is easy to see that the determinacy of Borel Gale-Stewart games implies the determinacy of Borel Wadge games. On the other hand, Louveau and Saint-Raymond have proved that this latter one is weaker than the first one, since it is already provable in second-order arithmetic, while the first one is not. It is also known that the determinacy of (effective) analytic Gale-Stewart games is equivalent to the determinacy of (effective) analytic Wadge games, see [11]. We prove in this paper that the determinacy of Wadge games between two players in charge of ω-languages accepted by 1-counter Büchi automata is equivalent to the (effective) analytic Wadge determinacy, and thus also equivalent to the existence of the real $0^\sharp$.

Then, using some recent results from [4] and some results of Set Theory, we prove that, (assuming ZFC is consistent), one can effectively construct a 1-counter Büchi automaton $\mathcal{A}$ and a Büchi automaton $\mathcal{B}$ such that: (1) There exists a model of ZFC in which Player 2 has a winning strategy in the Wadge game $W(L(\mathcal{A}), L(\mathcal{B}))$; (2) There exists a model of ZFC in which the Wadge game $W(L(\mathcal{A}), L(\mathcal{B}))$ is not determined. Moreover these are the only two possibilities, i.e. there are no models of ZFC in which Player 1 has a winning strategy in the Wadge game $W(L(\mathcal{A}), L(\mathcal{B}))$.

The paper is organized as follows. We recall some known notions in Section 2. We study context-free Gale-Stewart games in Section 3 and context-free Wadge games in Section 4. Some concluding remarks are given in Section 5.

2 Recall of some known notions

We assume the reader to be familiar with the theory of formal (ω-)languages [14, 13]. We recall the usual notations of formal language theory.

If Σ is a finite alphabet, a *non-empty finite word* over Σ is any sequence $x = a_1 \dots a_k$, where $a_i \in \Sigma$ for $i = 1, \dots, k$, and k is an integer ≥ 1. The *length* of x is k, denoted by $|x|$. The *empty word* has is denoted by λ; its length is 0. $\Sigma^\star$ is the *set of finite words* (including the empty word) over Σ. A (finitary) *language* V over an alphabet Σ is a subset of $\Sigma^\star$.

The *first infinite ordinal* is ω. An ω-*word* over Σ is an ω -sequence $a_1 \dots a_n \dots$, where for all integers $i \geq 1$, $\quad a_i \in \Sigma$. When $\sigma = a_1 \dots a_n \dots$ is an ω-word over Σ, we write $\sigma(n) = a_n$, $\sigma[n] = \sigma(1)\sigma(2) \dots \sigma(n)$ for all $n \geq 1$ and $\sigma[0] = \lambda$.

The usual concatenation product of two finite words u and v is denoted $u.v$ (and sometimes just uv). This product is extended to the product of a finite word u and an ω-word v: the infinite word $u.v$ is then the ω-word such that:

$(u.v)(k) = u(k)$ if $k \leq |u|$, and $(u.v)(k) = v(k - |u|)$ if $k > |u|$.

The *set of* ω-*words* over the alphabet Σ is denoted by Σ^ω. An ω-*language* V over an alphabet Σ is a subset of Σ^ω, and its complement (in Σ^ω) is $\Sigma^\omega - V$, denoted V^-.

The *prefix relation* is denoted $\sqsubseteq$: a finite word u is a *prefix* of a finite word v (respectively, an infinite word v), denoted $u \sqsubseteq v$, if and only if there exists a finite word w (respectively, an infinite word w), such that $v = u.w$.

If L is a finitary language (respectively, an ω-language) over the alphabet Σ then the set $\mathrm{Pref}(L)$ of prefixes of elements of L is defined by $\mathrm{Pref}(L) = \{u \in \Sigma^\star \mid \exists v \in L \;\; u \sqsubseteq v\}$.

We now recall the definition of k-counter Büchi automata which will be useful in the sequel.

Let k be an integer ≥ 1. A k-counter machine has k *counters*, each of which containing a non-negative integer. The machine can test whether the content of a given counter is zero or not. And transitions depend on the letter read by the machine, the current state of the finite control, and the tests about the values of the counters. Notice that in this model some λ-transitions are allowed. During these transitions the reading head of the machine does not move to the right, i.e. the machine does not read any more letter.

Formally a k-counter machine is a 4-tuple $\mathcal{M}$=(K, Σ, Δ, q_0), where K is a finite set of states, Σ is a finite input alphabet, $q_0 \in K$ is the initial state, and $\Delta \subseteq K \times (\Sigma \cup \{\lambda\}) \times \{0,1\}^k \times K \times \{0,1,-1\}^k$ is the transition relation. The k-counter machine $\mathcal{M}$ is said to be *real time* iff: $\Delta \subseteq K \times \Sigma \times \{0,1\}^k \times K \times \{0,1,-1\}^k$, i.e. iff there are no λ-transitions.

If the machine $\mathcal{M}$ is in state q and $c_i \in \mathbf{N}$ is the content of the i^{th} counter $\mathcal{C}_i$ then the configuration (or global state) of $\mathcal{M}$ is the $(k+1)$-tuple $(q, c_1, \dots, c_k)$.

For $a \in \Sigma \cup \{\lambda\}$, $q, q' \in K$ and $(c_1, \dots, c_k) \in \mathbf{N}^k$ such that $c_j = 0$ for $j \in E \subseteq \{1, \dots, k\}$ and $c_j > 0$ for $j \notin E$, if $(q, a, i_1, \dots, i_k, q', j_1, \dots, j_k) \in \Delta$ where $i_j = 0$ for $j \in E$ and $i_j = 1$ for $j \notin E$, then we write:

$$a : (q, c_1, \dots, c_k) \mapsto_{\mathcal{M}} (q', c_1 + j_1, \dots, c_k + j_k).$$

Thus the transition relation must obviously satisfy:
if $(q, a, i_1, \dots, i_k, q', j_1, \dots, j_k) \in \Delta$ and $i_m = 0$ for some $m \in \{1, \dots, k\}$ then $j_m = 0$ or $j_m = 1$ (but j_m may not be equal to -1).

Let $\sigma = a_1 a_2 \dots a_n \dots$ be an ω-word over Σ. An ω-sequence of configurations $r = (q_i, c_1^i, \dots c_k^i)_{i \geq 1}$ is called a run of $\mathcal{M}$ on σ iff:

(1) $(q_1, c_1^1, \dots c_k^1) = (q_0, 0, \dots, 0)$

(2) for each $i \geq 1$, there exists $b_i \in \Sigma \cup \{\lambda\}$ such that $b_i : (q_i, c_1^i, \dots c_k^i) \mapsto_{\mathcal{M}} (q_{i+1}, c_1^{i+1}, \dots c_k^{i+1})$ and such that $\quad a_1 a_2 \dots a_n \dots = b_1 b_2 \dots b_n \dots$

For every such run r, $\mathrm{In}(r)$ is the set of all states entered infinitely often during r.

▶ **Definition 1.** A Büchi k-counter automaton is a 5-tuple $\mathcal{M}=(K,\Sigma,\ \Delta,q_0,F)$, where $\mathcal{M}'=(K,\Sigma,\ \Delta,q_0)$ is a k-counter machine and $F\subseteq K$ is the set of accepting states. The ω-language accepted by $\mathcal{M}$ is:

$$L(\mathcal{M})=\{\sigma\in\Sigma^\omega\mid \text{ there exists a run r of } \mathcal{M} \text{ on } \sigma \text{ such that } \mathrm{In}(r)\cap F\neq\emptyset\}$$

The class of ω-languages accepted by Büchi k-counter automata is denoted $\mathbf{BCL}(k)_\omega$. The class of ω-languages accepted by *real time* Büchi k-counter automata will be denoted $\mathbf{r\text{-}BCL}(k)_\omega$. The class $\mathbf{BCL}(1)_\omega$ is a strict subclass of the class $\mathbf{CFL}_\omega$ of context free ω-languages accepted by Büchi pushdown automata.

We assume the reader to be familiar with basic notions of topology which may be found in [9, 10, 14, 13]. There is a natural metric on the set Σ^ω of infinite words over a finite alphabet Σ containing at least two letters which is called the *prefix metric* and is defined as follows. For $u,v\in\Sigma^\omega$ and $u\neq v$ let $\delta(u,v)=2^{-l_{\mathrm{pref}(u,v)}}$ where $l_{\mathrm{pref}(u,v)}$ is the first integer n such that the $(n+1)^{st}$ letter of u is different from the $(n+1)^{st}$ letter of v. This metric induces on Σ^ω the usual Cantor topology in which the *open subsets* of Σ^ω are of the form $W.\Sigma^\omega$, for $W\subseteq\Sigma^\star$. A set $L\subseteq\Sigma^\omega$ is a *closed set* iff its complement $\Sigma^\omega-L$ is an open set.

For $V\subseteq\Sigma^\star$ we denote $\mathrm{Lim}(V)=\{x\in\Sigma^\omega\mid\exists^\infty n\geq 1\ \ x[n]\in V\}$ the set of infinite words over Σ having infinitely many prefixes in V. Then the topological closure $\mathrm{Cl}(L)$ of a set $L\subseteq\Sigma^\omega$ is equal to $\mathrm{Lim}(\mathrm{Pref}(L))$. Thus we have also the following characterization of closed subsets of Σ^ω: a set $L\subseteq\Sigma^\omega$ is a closed subset of the Cantor space Σ^ω iff $L=\mathrm{Lim}(\mathrm{Pref}(L))$.

We now recall the definition of the *Borel Hierarchy* of subsets of X^ω.

▶ **Definition 2.** For a non-null countable ordinal α, the classes $\mathbf{\Sigma}^0_\alpha$ and $\mathbf{\Pi}^0_\alpha$ of the Borel Hierarchy on the topological space X^ω are defined as follows: $\mathbf{\Sigma}^0_1$ is the class of open subsets of X^ω, $\mathbf{\Pi}^0_1$ is the class of closed subsets of X^ω, and for any countable ordinal $\alpha\geq 2$:
$\mathbf{\Sigma}^0_\alpha$ is the class of countable unions of subsets of X^ω in $\bigcup_{\gamma<\alpha}\mathbf{\Pi}^0_\gamma$.
$\mathbf{\Pi}^0_\alpha$ is the class of countable intersections of subsets of X^ω in $\bigcup_{\gamma<\alpha}\mathbf{\Sigma}^0_\gamma$.

A set $L\subseteq X^\omega$ is Borel iff it is in the union $\bigcup_{\alpha<\omega_1}\mathbf{\Sigma}^0_\alpha=\bigcup_{\alpha<\omega_1}\mathbf{\Pi}^0_\alpha$, where ω_1 is the first uncountable ordinal.

There are also some subsets of X^ω which are not Borel. In particular the class of Borel subsets of X^ω is strictly included into the class $\mathbf{\Sigma}^1_1$ of *analytic sets* which are obtained by projection of Borel sets. The *co-analytic sets* are the complements of analytic sets.

▶ **Definition 3.** A subset A of X^ω is in the class $\mathbf{\Sigma}^1_1$ of *analytic* sets iff there exist a finite alphabet Y and a Borel subset B of $(X\times Y)^\omega$ such that $x\in A\leftrightarrow\exists y\in Y^\omega$ such that $(x,y)\in B$, where (x,y) is the infinite word over the alphabet $X\times Y$ such that $(x,y)(i)=(x(i),y(i))$ for each integer $i\geq 1$.

We now recall the notion of completeness with regard to reduction by continuous functions. For a countable ordinal $\alpha\geq 1$, a set $F\subseteq X^\omega$ is said to be a $\mathbf{\Sigma}^0_\alpha$ (respectively, $\mathbf{\Pi}^0_\alpha$, $\mathbf{\Sigma}^1_1$)-*complete set* iff for any set $E\subseteq Y^\omega$ (with Y a finite alphabet): $E\in\mathbf{\Sigma}^0_\alpha$ (respectively, $E\in\mathbf{\Pi}^0_\alpha$, $E\in\mathbf{\Sigma}^1_1$) iff there exists a continuous function $f:Y^\omega\to X^\omega$ such that $E=f^{-1}(F)$.

We now recall the definition of classes of the arithmetical hierarchy of ω-languages, see [14]. Let X be a finite alphabet. An ω-language $L\subseteq X^\omega$ belongs to the class Σ_n if and only if there exists a recursive relation $R_L\subseteq(\mathbb{N})^{n-1}\times X^\star$ such that:
$L=\{\sigma\in X^\omega\mid\exists a_1\ldots Q_n a_n\quad(a_1,\ldots,a_{n-1},\sigma[a_n+1])\in R_L\}$,

where Q_i is one of the quantifiers $\forall$ or $\exists$ (not necessarily in an alternating order). An ω-language $L \subseteq X^\omega$ belongs to the class Π_n if and only if its complement $X^\omega - L$ belongs to the class Σ_n. The class Σ^1_1 is the class of *effective analytic sets* which are obtained by projection of arithmetical sets. An ω-language $L \subseteq X^\omega$ belongs to the class Σ^1_1 if and only if there exists a recursive relation $R_L \subseteq \mathbb{N} \times \{0,1\}^\star \times X^\star$ such that:

$$L = \{\sigma \in X^\omega \mid \exists \tau(\tau \in \{0,1\}^\omega \wedge \forall n \exists m((n, \tau[m], \sigma[m]) \in R_L))\}.$$

Then an ω-language $L \subseteq X^\omega$ is in the class Σ^1_1 iff it is the projection of an ω-language over the alphabet $X \times \{0,1\}$ which is in the class Π_2. The class Π^1_1 of *effective co-analytic sets* is simply the class of complements of effective analytic sets.

Recall that the (lightface) class Σ^1_1 of effective analytic sets is strictly included into the (boldface) class $\mathbf{\Sigma}^1_1$ of analytic sets.

Recall that a Büchi Turing machine is just a Turing machine working on infinite inputs with a Büchi-like acceptance condition, and that the class of ω-languages accepted by Büchi Turing machines is the class Σ^1_1 of effective analytic sets [2, 14]. On the other hand, one can construct, using a classical construction (see for instance [7]), from a Büchi Turing machine $\mathcal{T}$, a 2-counter Büchi automaton $\mathcal{A}$ accepting the same ω-language. Thus one can state the following proposition.

▶ **Proposition 4.** *An ω-language $L \subseteq X^\omega$ is in the class Σ^1_1 iff it is accepted by a non deterministic Büchi Turing machine, hence iff it is in the class* $\mathbf{BCL}(2)_\omega$.

3 Context-free Gale-Stewart games

We first recall the definition of Gale-Stewart games.

▶ **Definition 5** ([8])**.** Let $A \subseteq X^\omega$, where X is a finite alphabet. The Gale-Stewart game $G(A)$ is a game with perfect information between two players. Player 1 first writes a letter $a_1 \in X$, then Player 2 writes a letter $b_1 \in X$, then Player 1 writes $a_2 \in X$, and so on ... After ω steps, the two players have composed a word $x = a_1 b_1 a_2 b_2 \ldots$ of X^ω. Player 1 wins the play iff $x \in A$, otherwise Player 2 wins the play.

Let $A \subseteq X^\omega$ and $G(A)$ be the associated Gale-Stewart game. A strategy for Player 1 is a function $F_1 : (X^2)^\star \to X$ and a strategy for Player 2 is a function $F_2 : (X^2)^\star X \to X$. Player 1 follows the strategy F_1 in a play if for each integer $n \geq 1 \quad a_n = F_1(a_1 b_1 a_2 b_2 \cdots a_{n-1} b_{n-1})$. If Player 1 wins every play in which she has followed the strategy F_1, then we say that the strategy F_1 is a winning strategy (w.s.) for Player 1. The notion of winning strategy for Player 2 is defined in a similar manner.

The game $G(A)$ is said to be determined if one of the two players has a winning strategy.

We shall denote $\mathbf{Det}(\mathcal{C})$, where $\mathcal{C}$ is a class of ω-languages, the sentence : "Every Gale-Stewart game $G(A)$, where $A \subseteq X^\omega$ is an ω-language in the class $\mathcal{C}$, is determined".

Notice that, in the whole paper, we assume that ZFC is consistent, and all results, lemmas, propositions, theorems, are stated in ZFC unless we explicitely give another axiomatic framework.

We can now state our first result.

▶ **Proposition 6.** $\mathbf{Det}(\Sigma^1_1) \iff \mathbf{Det}(\mathbf{r\text{-}BCL}(8)_\omega)$.

Proof. The implication $\mathbf{Det}(\Sigma^1_1) \implies \mathbf{Det}(\mathbf{r\text{-}BCL}(8)_\omega)$ is obvious since $\mathbf{r\text{-}BCL}(8)_\omega \subseteq \Sigma^1_1$.

To prove the reverse implication, we assume that $\mathbf{Det}(\mathbf{r\text{-}BCL}(8)_\omega)$ holds and we show that every Gale-Stewart game $G(A)$, where $A \subseteq X^\omega$ is an ω-language in the class Σ^1_1, or equivalently in the class $\mathbf{BCL}(2)_\omega$ by Proposition 4, is determined.

Let then $L \subseteq \Sigma^\omega$, where Σ is a finite alphabet, be an ω-language in the class $\mathbf{BCL}(2)_\omega$.

Let E be a new letter not in Σ, S be an integer ≥ 1, and $\theta_S : \Sigma^\omega \to (\Sigma \cup \{E\})^\omega$ be the function defined, for all $x \in \Sigma^\omega$, by:

$$\theta_S(x) = x(1).E^S.x(2).E^{S^2}.x(3).E^{S^3}.x(4)\dots x(n).E^{S^n}.x(n+1).E^{S^{n+1}}\dots$$

We proved in [3] that if $k = \mathit{cardinal}(\Sigma) + 2$, $S \geq (3k)^3$ is an integer, then one can effectively construct from a Büchi 2-counter automaton $\mathcal{A}_1$ accepting L a real time Büchi 8-counter automaton $\mathcal{A}_2$ such that $L(\mathcal{A}_2) = \theta_S(L)$. In the sequel we assume that we have fixed an integer $S \geq (3k)^3$ which is *even*.

Notice that the set $\theta_S(\Sigma^\omega)$ is a closed subset of the Cantor space Σ^ω. An ω-word $x \in (\Sigma \cup \{E\})^\omega$ is in $\theta_S(\Sigma^\omega)^-$ iff it has one prefix which is not in $\mathrm{Pref}(\theta_S(\Sigma^\omega))$. Let $L' \subseteq (\Sigma \cup \{E\})^\omega$ be the set of ω-words $y \in (\Sigma \cup \{E\})^\omega$ for which there is an integer $n \geq 1$ such that $y[2n-1] \in \mathrm{Pref}(\theta_S(\Sigma^\omega))$ and $y[2n] \notin \mathrm{Pref}(\theta_S(\Sigma^\omega))$. It is easy to see that L' is accepted by a real time Büchi 2-counter automaton.

The class $\mathbf{r\text{-}BCL}(8)_\omega \supseteq \mathbf{r\text{-}BCL}(2)_\omega$ is closed under finite union in an effective way, so $\theta_S(L) \cup L'$ is accepted by a real time Büchi 8-counter automaton $\mathcal{A}_3$ which can be effectively constructed from $\mathcal{A}_2$.

As we have assumed that $\mathbf{Det}(\mathbf{r\text{-}BCL}(8)_\omega)$ holds, the game $G(\theta_S(L) \cup L')$ is determined, i.e. one of the two players has a w.s. in the game $G(\theta_S(L) \cup L')$. We now show that the game $G(L)$ is itself determined.

We shall say that, during an infinite play, Player 1 "goes out" of the *closed* set $\theta_S(\Sigma^\omega)$ if the final play y composed by the two players has a prefix $y[2n] \in \mathrm{Pref}(\theta_S(\Sigma^\omega))$ such that $y[2n+1] \notin \mathrm{Pref}(\theta_S(\Sigma^\omega))$. We define in a similar way the sentence "Player 2 goes out of the *closed* set $\theta_S(\Sigma^\omega)$".

Assume first that Player 1 has a w.s. F_1 in the game $G(\theta_S(L) \cup L')$. Then Player 1 never "goes out" of the set $\theta_S(\Sigma^\omega)$ when she follows this w.s. because otherwise the final play y composed by the two players has a prefix $y[2n] \in \mathrm{Pref}(\theta_S(\Sigma^\omega))$ such that $y[2n+1] \notin \mathrm{Pref}(\theta_S(\Sigma^\omega))$ and thus $y \notin \theta_S(L) \cup L'$. Consider now a play in which Player 2 does not go out of $\theta_S(\Sigma^\omega)$. If player 1 follows her w.s. F_1 then the two players remain in the set $\theta_S(\Sigma^\omega)$. But we have fixed S to be an **even** integer. So the two players compose an ω-word

$$\theta_S(x) = x(1).E^S.x(2).E^{S^2}.x(3).E^{S^3}.x(4)\dots x(n).E^{S^n}.x(n+1).E^{S^{n+1}}\dots$$

and the letters $x(k)$ are written by player 1 for k an odd integer and by Player 2 for k an even integer because S is even. Moreover Player 1 wins the play iff the ω-word $x(1)x(2)x(3)\dots x(n)\dots$ is in L. This implies that Player 1 has also a w.s. in the game $G(L)$.

Assume now that Player 2 has a w.s. F_2 in the game $G(\theta_S(L) \cup L')$. Then Player 2 never "goes out" of the set $\theta_S(\Sigma^\omega)$ when he follows this w.s. because otherwise the final play y composed by the two players has a prefix $y[2n-1] \in \mathrm{Pref}(\theta_S(\Sigma^\omega))$ such that $y[2n] \notin \mathrm{Pref}(\theta_S(\Sigma^\omega))$ and thus $y \in L'$ hence also $y \in \theta_S(L) \cup L'$. Consider now a play in which Player 1 does not go out of $\theta_S(\Sigma^\omega)$. If player 2 follows his w.s. F_2 then the two players remain in the set $\theta_S(\Sigma^\omega)$. So the two players compose an ω-word

$$\theta_S(x) = x(1).E^S.x(2).E^{S^2}.x(3).E^{S^3}.x(4)\dots x(n).E^{S^n}.x(n+1).E^{S^{n+1}}\dots$$

where the letters $x(k)$ are written by player 1 for k an odd integer and by Player 2 for k an even integer. Moreover Player 2 wins the play iff the ω-word $x(1)x(2)x(3)\dots x(n)\dots$ is not in L. This implies that Player 2 has also a w.s. in the game $G(L)$. ◀

▶ **Theorem 7.** $\mathbf{Det}(\Sigma^1_1) \Longleftrightarrow \mathbf{Det}(\mathbf{CFL}_\omega) \Longleftrightarrow \mathbf{Det}(\mathbf{BCL}(1)_\omega)$.

Proof. The implications $\mathbf{Det}(\Sigma^1_1) \Longrightarrow \mathbf{Det}(\mathbf{CFL}_\omega) \Longrightarrow \mathbf{Det}(\mathbf{BCL}(1)_\omega)$ are obvious since $\mathbf{BCL}(1)_\omega \subseteq \mathbf{CFL}_\omega \subseteq \Sigma^1_1$.

To prove the reverse implication $\mathbf{Det}(\mathbf{BCL}(1)_\omega) \Longrightarrow \mathbf{Det}(\Sigma^1_1)$, we assume that $\mathbf{Det}(\mathbf{BCL}(1)_\omega)$ holds and we are going to show that then every Gale-Stewart game $G(L)$, where $L \subseteq X^\omega$ is an ω-language in the class $\mathbf{r}\text{-}\mathbf{BCL}(8)_\omega$ is determined. Then Proposition 6 will imply that $\mathbf{Det}(\Sigma^1_1)$ also holds. Let then $L(\mathcal{A}) \subseteq \Gamma^\omega$, where Γ is a finite alphabet and $\mathcal{A}$ is a real time Büchi 8-counter automaton.

We now recall the following coding which was used in the paper [3].

Let K be the product of the eight first prime numbers. An ω-word $x \in \Gamma^\omega$ was coded by the ω-word

$$h_K(x) = A.C^K.x(1).B.C^{K^2}.A.C^{K^2}.x(2).B.C^{K^3}.A.C^{K^3}.x(3).B\dots B.C^{K^n}.A.C^{K^n}.x(n).B\dots$$

over the alphabet $\Gamma_1 = \Gamma \cup \{A, B, C\}$, where A, B, C are new letters not in Γ. We are going to use here a slightly different coding which we now define. Let then

$$h(x) = C^K.C.A.x(1).C^{K^2}.A.C^{K^2}.C.x(2).B.C^{K^3}.A.C^{K^3}.C.A.x(3)\dots$$
$$\dots C^{K^{2n}}.A.C^{K^{2n}}.C.x(2n).B.C^{K^{2n+1}}.A.C^{K^{2n+1}}.C.A.x(2n+1)\dots$$

We now explain the rules used to obtain the ω-word $h(x)$ from the ω-word $h_K(x)$.

(1) The first letter A of the word $h_K(x)$ has been suppressed.

(2) The letters B following a letter $x(2n+1)$, for $n \geq 1$, have been suppressed.

(3) A letter C has been added before each letter $x(2n)$, for $n \geq 1$.

(4) A block of two letters $C.A$ has been added before each letter $x(2n+1)$, for $n \geq 1$.

The reasons behind this changes are the following ones. Assume that two players alternatively write letters from the alphabet $\Gamma_1 = \Gamma \cup \{A, B, C\}$ and that they finally produce an ω-word in the form $h(x)$. Due to the above changes we have now the two following properties which will be useful in the sequel.

(1) The letters $x(2n+1)$, for $n \geq 0$, have been written by Player 1, and the letters $x(2n)$, for $n \geq 1$, have been written by Player 2.

(2) After a sequence of consecutive letters C, the first letter which is not a C has always been written by Player 2.

We proved in [3] that, from a real time Büchi 8-counter automaton $\mathcal{A}$ accepting $L(\mathcal{A}) \subseteq \Gamma^\omega$, one can effectively construct a Büchi 1-counter automaton $\mathcal{A}_1$ accepting the ω-language $h_K(L(\mathcal{A})) \cup h_K(\Gamma^\omega)^-$. We can easily check that the changes in $h_K(x)$ leading to the coding $h(x)$ have no influence with regard to the proof of this result in [3] and thus one can also effectively construct a Büchi 1-counter automaton $\mathcal{A}_2$ accepting the ω-language $h(L(\mathcal{A})) \cup h(\Gamma^\omega)^-$.

On the other hand we can remark that all ω-words in the form $h(x)$ belong to the ω-language $H \subseteq (\Gamma_1)^\omega$ of ω-words y of the following form:

$$y = C^{n_1}.C.A.x(1).C^{n_2}.A.C^{n'_2}.C.x(2).B.C^{n_3}.A.C^{n'_3}.C.A.x(3)\dots$$
$$\dots C^{n_{2n}}.A.C^{n'_{2n}}.C.x(2n).B.C^{n_{2n+1}}.A.C^{n'_{2n+1}}.C.A.x(2n+1)\dots$$

where for all integers $i \geq 1$ the letters $x(i)$ belong to Γ and the n_i, n'_i, are even non-null integers.

An important fact is the following property of H which extends the same property of the set $h(\Gamma^\omega)$. Assume that two players alternatively write letters from the alphabet $\Gamma_1 = \Gamma \cup \{A, B, C\}$ and that they finally produce an ω-word y in H in the above form. Then we have the two following facts:

(1) The letters $x(2n+1)$, for $n \geq 0$, have been written by Player 1, and the letters $x(2n)$, for $n \geq 1$, have been written by Player 2.

(2) After a sequence of consecutive letters C, the first letter which is not a C has always been written by Player 2.

Let now $V = \mathrm{Pref}(H) \cap (\Gamma_1)^\star.C$. So a finite word over the alphabet Γ_1 is in V iff it is a prefix of some word in H and its last letter is a C. It is easy to see that the topological closure of H is

$$\mathrm{Cl}(H) = H \ \cup \ V.C^\omega.$$

Notice that an ω-word in $\mathrm{Cl}(H)$ is not in $h(\Gamma^\omega)$ iff a sequence of consecutive letters C has not the good length. Thus if two players alternatively write letters from the alphabet Γ_1 and produce an ω-word $y \in \mathrm{Cl}(H) - h(\Gamma^\omega)$ then it is Player 2 who has gone out of the set $h(\Gamma^\omega)$ at some step of the play. This will be important in the sequel.

It is very easy to see that the ω-language H is regular and to construct a Büchi automaton $\mathcal{H}$ accepting it. Moreover it is known that the class $\mathbf{BCL}(1)_\omega$ is effectively closed under intersection with regular ω-languages (this can be seen using a classical construction of a product automaton). Thus one can also construct a Büchi 1-counter automaton $\mathcal{A}_3$ accepting the ω-language $h(L(\mathcal{A}))\cup[h(\Gamma^\omega)^- \cap H]$.

We denote also U the set of finite words u over Γ_1 such that $|u| = 2n$ for some integer $n \geq 1$ and $u[2n-1] \in \mathrm{Pref}(H)$ and $u = u[2n] \notin \mathrm{Pref}(H)$.

Now we set:

$$\mathcal{L} \ = \ h(L(\mathcal{A})) \ \cup \ [h(\Gamma^\omega)^- \cap H] \ \cup \ V.C^\omega \ \cup \ U.(\Gamma_1)^\omega$$

We have already seen that the ω-language $h(L(\mathcal{A}))\cup[h(\Gamma^\omega)^- \cap H]$ is accepted by a Büchi 1-counter automaton $\mathcal{A}_3$. On the other hand the ω-language H is regular and it is accepted by a Büchi automaton $\mathcal{H}$. Thus the finitary language $\mathrm{Pref}(H)$ is also regular, the languages U and V are also regular, and the ω-languages $V.C^\omega$ and $U.(\Gamma_1)^\omega$ are regular. This implies that one can construct a Büchi 1-counter automaton $\mathcal{A}_4$ accepting the language $\mathcal{L}$.

By hypothesis we assume that $\mathbf{Det}(\mathbf{BCL}(1)_\omega)$ holds and thus the game $G(\mathcal{L})$ is determined. We are going to show that this implies that the game $G(L(\mathcal{A}))$ itself is determined.

Assume firstly that Player 1 has a winning strategy F_1 in the game $G(\mathcal{L})$.

If during an infinite play, the two players compose an infinite word z, and Player 2 "does not go out of the set $h(\Gamma^\omega)$" then we claim that also Player 1, following her strategy F_1, "does not go out of the set $h(\Gamma^\omega)$". Indeed if Player 1 goes out of the set $h(\Gamma^\omega)$ then due to the above remark this would imply that Player 1 also goes out of the set $\mathrm{Cl}(H)$: there is an integer $n \geq 0$ such that $z[2n] \in \mathrm{Pref}(H)$ but $z[2n+1] \notin \mathrm{Pref}(H)$. So $z \notin h(L(\mathcal{A})) \cup [h(\Gamma^\omega)^- \cap H] \cup V.C^\omega$. Moreover it follows from the definition of U that $z \notin U.(\Gamma_1)^\omega$. Thus If Player 1 goes out of the set $h(\Gamma^\omega)$ then she looses the game.

Consider now an infinite play in which Player 2 "does not go out of the set $h(\Gamma^\omega)$". Then Player 1, following her strategy F_1, "does not go out of the set $h(\Gamma^\omega)$". Thus the two players write an infinite word $z = h(x)$ for some infinite word $x \in \Gamma^\omega$. But the letters $x(2n+1)$, for $n \geq 0$, have been written by Player 1, and the letters $x(2n)$, for $n \geq 1$, have been written by Player 2. Player 1 wins the play iff $x \in L(\mathcal{A})$ and Player 1 wins always the play when she uses her strategy F_1. This implies that Player 1 has also a w.s. in the game $G(L(\mathcal{A}))$.

Assume now that Player 2 has a winning strategy F_2 in the game $G(\mathcal{L})$.

If during an infinite play, the two players compose an infinite word z, and Player 1 "does not go out of the set $h(\Gamma^\omega)$" then we claim that also Player 2, following his strategy F_2, "does not go out of the set $h(\Gamma^\omega)$". Indeed if Player 2 goes out of the set $h(\Gamma^\omega)$ and the final play z remains in $\mathrm{Cl}(H)$ then $z \in [h(\Gamma^\omega)^- \cap H] \cup V.C^\omega \subseteq \mathcal{L}$ and Player 2 looses. If Player 1 does not go out of the set $\mathrm{Cl}(H)$ and at some step of the play, Player 2 goes out of $\mathrm{Pref}(H)$, i.e. there is an integer $n \geq 1$ such that $z[2n-1] \in \mathrm{Pref}(H)$ and $z[2n] \notin \mathrm{Pref}(H)$, then $z \in U.(\Gamma_1)^\omega \subseteq \mathcal{L}$ and Player 2 looses.

Assume now that Player 1 "does not go out of the set $h(\Gamma^\omega)$". Then Player 2 follows his w. s. F_2, and then "never goes out of the set $h(\Gamma^\omega)$". Thus the two players write an infinite word $z = h(x)$ for some infinite word $x \in \Gamma^\omega$. But the letters $x(2n+1)$, for $n \geq 0$, have been written by Player 1, and the letters $x(2n)$, for $n \geq 1$, have been written by Player 2. Player 2 wins the play iff $x \notin L(\mathcal{A})$ and Player 2 wins always the play when he uses his strategy F_2. This implies that Player 2 has also a w.s. in the game $G(L(\mathcal{A}))$. ◀

Looking carefully at the above proof, we can obtain a stronger result:

▶ **Theorem 8.** $\mathbf{Det}(\Sigma^1_1) \iff \mathbf{Det}(\mathbf{CFL}_\omega) \iff \mathbf{Det}(\mathbf{r\text{-}BCL}(1)_\omega)$.

4 Context-free Wadge games

We first recall the notion of Wadge games.

▶ **Definition 9** (Wadge [17]). Let $L \subseteq X^\omega$ and $L' \subseteq Y^\omega$. The Wadge game $W(L, L')$ is a game with perfect information between two players, Player 1 who is in charge of L and Player 2 who is in charge of L'. Player 1 first writes a letter $a_1 \in X$, then Player 2 writes a letter $b_1 \in Y$, then Player 1 writes a letter $a_2 \in X$, and so on. The two players alternatively write letters a_n of X for Player 1 and b_n of Y for Player 2. After ω steps, Player 1 has written an ω-word $a \in X^\omega$ and Player 2 has written an ω-word $b \in Y^\omega$. Player 2 is allowed to skip, even infinitely often, provided he really writes an ω-word in ω steps. Player 2 wins the play iff $[a \in L \leftrightarrow b \in L']$, i.e. iff: $[(a \in L \text{ and } b \in L') \text{ or } (a \notin L \text{ and } b \notin L' \text{ and } b \text{ is infinite})]$.

Recall that a strategy for Player 1 is a function $\sigma : (Y \cup \{s\})^\star \to X$. And a strategy for Player 2 is a function $f : X^+ \to Y \cup \{s\}$. The strategy σ is a winning strategy for Player 1 iff she always wins a play when she uses the strategy σ, i.e. when the n^{th} letter she writes is given by $a_n = \sigma(b_1 \dots b_{n-1})$, where b_i is the letter written by Player 2 at step i and $b_i = s$ if Player 2 skips at step i. A winning strategy for Player 2 is defined in a similar manner.

The game $W(L, L')$ is said to be determined if one of the two players has a winning strategy. In the sequel we shall denote $\mathbf{W\text{-}Det}(\mathcal{C})$, where $\mathcal{C}$ is a class of ω-languages, the sentence: "All Wadge games $W(L, L')$, where $L \subseteq X^\omega$ and $L' \subseteq Y^\omega$ are ω-languages in the class $\mathcal{C}$, are determined".

There is a close relationship between Wadge reducibility and games.

▶ **Definition 10** (Wadge [17]). Let X, Y be two finite alphabets. For $L \subseteq X^\omega$ and $L' \subseteq Y^\omega$, L is said to be Wadge reducible to L' ($L \leq_W L'$) iff there exists a continuous function $f : X^\omega \to Y^\omega$, such that $L = f^{-1}(L')$. L and L' are Wadge equivalent iff $L \leq_W L'$ and $L' \leq_W L$. This will be denoted by $L \equiv_W L'$. And we shall say that $L <_W L'$ iff $L \leq_W L'$ but not $L' \leq_W L$.
The relation $\leq_W$ is reflexive and transitive, and $\equiv_W$ is an equivalence relation.
The *equivalence classes* of $\equiv_W$ are called *Wadge degrees*.

▶ **Theorem 11** (Wadge). *Let $L \subseteq X^\omega$ and $L' \subseteq Y^\omega$ where X and Y are finite alphabets. Then $L \leq_W L'$ if and only if Player 2 has a winning strategy in the Wadge game $W(L, L')$.*

The Wadge hierarchy WH is the class of Borel subsets of a set X^ω, where X is a finite set, equipped with $\leq_W$ and with $\equiv_W$. Using Wadge games, Wadge proved that, up to the complement and $\equiv_W$, it is a well ordered hierarchy which provides a great refinement of the Borel hierarchy.

We can now state the following result on determinacy of context-free Wadge games.

▶ **Theorem 12.** $\mathbf{Det}(\Sigma^1_1) \iff \mathbf{W\text{-}Det}(\mathbf{CFL}_\omega) \iff \mathbf{W\text{-}Det}(\mathbf{BCL}(1)_\omega) \iff \mathbf{W\text{-}Det}(\mathbf{r\text{-}BCL}(1)_\omega)$.

Recall that, assuming that ZFC is consistent, there are some models of ZFC in which $\mathbf{Det}(\Sigma^1_1)$ does not hold. Therefore there are some models of ZFC in which some Wadge games $W(L(\mathcal{A}), L(\mathcal{B}))$, where $\mathcal{A}$ and $\mathcal{B}$ are Büchi 1-counter automata, are not determined. We are going to prove that this may be also the case when $\mathcal{B}$ is a Büchi automaton (without counter). To prove this, we use a recent result of [4] and some results of set theory, so we now briefly recall some notions of set theory and refer the reader to [4] and to a textbook like [8] for more background on set theory.

The usual axiomatic system ZFC is Zermelo-Fraenkel system ZF plus the axiom of choice AC. The axioms of ZFC express some natural facts that we consider to hold in the universe of sets. A model $(\mathbf{V}, \in)$ of an arbitrary set of axioms $\mathbb{A}$ is a collection $\mathbf{V}$ of sets, equipped with the membership relation $\in$, where "$x \in y$" means that the set x is an element of the set y, which satisfies the axioms of $\mathbb{A}$. We often say " the model $\mathbf{V}$" instead of "the model $(\mathbf{V}, \in)$".

We say that two sets A and B have same cardinality iff there is a bijection from A onto B and we denote this by $A \approx B$. The relation $\approx$ is an equivalence relation. Using the axiom of choice AC, one can prove that any set A can be well-ordered so there is an ordinal γ such that $A \approx \gamma$. In set theory the cardinal of the set A is then formally defined as the smallest such ordinal γ. The infinite cardinals are usually denoted by $\aleph_0, \aleph_1, \aleph_2, \ldots, \aleph_\alpha, \ldots$ The continuum hypothesis CH says that the first uncountable cardinal $\aleph_1$ is equal to $2^{\aleph_0}$ which is the cardinal of the continuum.

If $\mathbf{V}$ is a model of ZF and $\mathbf{L}$ is the class of *constructible sets* of $\mathbf{V}$, then the class $\mathbf{L}$ is a model of ZFC + CH. Notice that the axiom V=L, which means "every set is constructible", is consistent with ZFC because $\mathbf{L}$ is a model of ZFC + V=L.

Consider now a model $\mathbf{V}$ of ZFC and the class of its constructible sets $\mathbf{L} \subseteq \mathbf{V}$ which is another model of ZFC. It is known that the ordinals of $\mathbf{L}$ are also the ordinals of $\mathbf{V}$, but the cardinals in $\mathbf{V}$ may be different from the cardinals in $\mathbf{L}$. In particular, the first uncountable cardinal in $\mathbf{L}$ is denoted $\aleph_1^{\mathbf{L}}$, and it is in fact an ordinal of $\mathbf{V}$ which is denoted $\omega_1^{\mathbf{L}}$. It is well-known that in general this ordinal satisfies the inequality $\omega_1^{\mathbf{L}} \leq \omega_1$. In a model $\mathbf{V}$ of

the axiomatic system ZFC + V=L the equality $\omega_1^{\mathbf{L}} = \omega_1$ holds, but in some other models of ZFC the inequality may be strict and then $\omega_1^{\mathbf{L}} < \omega_1$.

The following result was proved in [4].

▶ **Theorem 13.** *There exists a real-time 1-counter Büchi automaton $\mathcal{A}$, which can be effectively constructed, such that the topological complexity of the ω-language $L(\mathcal{A})$ is not determined by the axiomatic system* ZFC. *Indeed it holds that :*

(1) *(*ZFC + V=L*).* *The ω-language $L(\mathcal{A})$ is an analytic but non-Borel set.*

(2) *(*ZFC + $\omega_1^{\mathbf{L}} < \omega_1$*).* *The ω-language $L(\mathcal{A})$ is a $\mathbf{\Pi}_2^0$-set.*

We now state the following new result. To prove it we use in particular the above Theorem 13, the link between Wadge games and Wadge reducibility, the $\mathbf{\Pi}_2^0$-completeness of the regular ω-language $(0^\star.1)^\omega \subseteq \{0,1\}^\omega$, the Shoenfield's Absoluteness Theorem, and the notion of extensions of a model of ZFC.

▶ **Theorem 14.** [1] *Let $\mathcal{B}$ be a Büchi automaton accepting the regular ω-language $(0^\star.1)^\omega \subseteq \{0,1\}^\omega$. Then one can effectively construct a real-time 1-counter Büchi automaton $\mathcal{A}$ such that:*

(1) *(*ZFC + $\omega_1^{\mathbf{L}} < \omega_1$*). Player 2 has a winning strategy F in the Wadge game $W(L(\mathcal{A}), L(\mathcal{B}))$. But F cannot be recursive and not even hyperarithmetical.*

(2) *(*ZFC + $\omega_1^{\mathbf{L}} = \omega_1$*). The Wadge game $W(L(\mathcal{A}), L(\mathcal{B}))$ is not determined.*

▶ Remark 15. Every model of ZFC is either a model of (ZFC + $\omega_1^{\mathbf{L}} < \omega_1$) or a model of (ZFC + $\omega_1^{\mathbf{L}} = \omega_1$). Thus there are no models of ZFC in which Player 1 has a winning strategy in the Wadge game $W(L(\mathcal{A}), L(\mathcal{B}))$.

▶ Remark 16. In order to prove Theorem 14 we do not need to use any large cardinal axiom or even the consistency of such an axiom, like the axiom of analytic determinacy.

5 Concluding remarks

We have proved that the determinacy of Gale-Stewart games whose winning sets are accepted by (real-time) 1-counter Büchi automata is equivalent to the determinacy of (effective) analytic Gale-Stewart games which is known to be a large cardinal assumption.

On the other hand we have proved a similar result about the determinacy of Wadge games. We have also obtained an amazing result, proving that one can effectively construct a real-time 1-counter Büchi automaton $\mathcal{A}$ and a Büchi automaton $\mathcal{B}$ such that the sentence "the Wadge game $W(L(\mathcal{A}), L(\mathcal{B}))$ is determined" is actually independent from ZFC.

Notice that it is still unknown whether the determinacy of Wadge games $W(L(\mathcal{A}), L(\mathcal{B}))$, where $\mathcal{A}$ and $\mathcal{B}$ are Muller tree automata (reading infinite labelled trees) , is provable within ZFC or needs some large cardinal assumptions to be proved.

Acknowledgements I wish to thank the anonymous referees for useful comments on a preliminary version of this paper.

[1] This result has been recently exposed in the Workshops GASICS 2010, Paris, September 2010, and GAMES 2010, Oxford, September 2010, but it has never been published.

References

1 A. Carayol, M. Hague, A. Meyer, C.-H. L. Ong, and O. Serre. Winning regions of higher-order pushdown games. In *Proceedings of the Twenty-Third Annual IEEE Symposium on Logic in Computer Science, LICS 2008, 24-27 June 2008, Pittsburgh, PA, USA*, pages 193–204. IEEE Computer Society, 2008.

2 R.S. Cohen and A.Y. Gold. ω-computations on Turing machines. *Theoretical Computer Science*, 6:1–23, 1978.

3 O. Finkel. Borel ranks and Wadge degrees of omega context free languages. *Mathematical Structures in Computer Science*, 16(5):813–840, 2006.

4 O. Finkel. The complexity of infinite computations in models of set theory. *Logical Methods in Computer Science*, 5(4:4):1–19, 2009.

5 O. Finkel. Highly undecidable problems for infinite computations. *Theoretical Informatics and Applications*, 43(2):339–364, 2009.

6 L. Harrington. Analytic determinacy and $0^\sharp$. *Journal of Symbolic Logic*, 43(4):685–693, 1978.

7 J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to automata theory, languages, and computation.* Addison-Wesley Publishing Co., Reading, Mass., 2001. Addison-Wesley Series in Computer Science.

8 T. Jech. *Set theory, third edition.* Springer, 2002.

9 A. S. Kechris. *Classical descriptive set theory.* Springer-Verlag, New York, 1995.

10 H. Lescow and W. Thomas. Logical specifications of infinite computations. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *A Decade of Concurrency*, volume 803 of *Lecture Notes in Computer Science*, pages 583–621. Springer, 1994.

11 A. Louveau and J. Saint-Raymond. The strength of Borel Wadge determinacy. In *Cabal Seminar 81–85*, volume 1333 of *Lecture Notes in Mathematics*, pages 1–30. Springer, 1988.

12 P.G. Odifreddi. *Classical Recursion Theory, Vol I*, volume 125 of *Studies in Logic and the Foundations of Mathematics.* North-Holland Publishing Co., Amsterdam, 1989.

13 D. Perrin and J.-E. Pin. *Infinite words, automata, semigroups, logic and games*, volume 141 of *Pure and Applied Mathematics.* Elsevier, 2004.

14 L. Staiger. ω-languages. In *Handbook of formal languages, Vol. 3*, pages 339–387. Springer, Berlin, 1997.

15 W. Thomas. On the synthesis of strategies in infinite games. In *Proceedings of the International Conference STACS 1995*, volume 900 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 1995.

16 W. Thomas. Church's problem and a tour through automata theory. In Arnon Avron, Nachum Dershowitz, and Alexander Rabinovich, editors, *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*, pages 635–655. Springer, 2008.

17 W. Wadge. *Reducibility and determinateness in the Baire space.* PhD thesis, University of California, Berkeley, 1983.

18 I. Walukiewicz. Pushdown processes: games and model checking. *Information and Computation*, 157:234–263, 2000.

The dimension of ergodic random sequences

Mathieu Hoyrup[1]

1 **LORIA, INRIA Nancy - Grand Est**
LORIA (Carte, bât. B)
615 rue du jardin botanique
BP 239
54506 Vandœuvre-lès-Nancy
France
`mathieu.hoyrup@loria.fr`

Abstract

Let μ be a computable ergodic shift-invariant measure over $\{0,1\}^{\mathbb{N}}$. Providing a constructive proof of Shannon-McMillan-Breiman theorem, V'yugin proved that if $x \in \{0,1\}^{\mathbb{N}}$ is Martin-Löf random w.r.t. μ then the strong effective dimension $\mathrm{Dim}(x)$ of x equals the entropy of μ. Whether its effective dimension $\dim(x)$ also equals the entropy was left as an open problem. In this paper we settle this problem, providing a positive answer. A key step in the proof consists in extending recent results on Birkhoff's ergodic theorem for Martin-Löf random sequences. At the same time, we present extensions of some previous results.

As pointed out by a referee the main result can also be derived from results by Hochman [8], using rather different considerations.

1998 ACM Subject Classification E.4 Coding and Information Theory, F.4.1 Mathematical Logic, G.3 Probability and Statistics

Keywords and phrases Shannon-McMillan-Breiman theorem; Martin-Löf random sequence; effective Hausdorff dimension; compression rate; entropy

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.567

1 Introduction

The *effective dimension* and *strong effective dimension* of an infinite binary sequence x are defined as

$$\dim(x) = \liminf_n \frac{K(x\restriction_n)}{n}$$
$$\mathrm{Dim}(x) = \limsup_n \frac{K(x\restriction_n)}{n},$$

where $K(w)$ is the Kolmogorov complexity of w.

They can be characterized as effective versions of Hausdorff and packing dimensions respectively, or by divergence of gales (see [12, 15, 1] for the original results and [13] for a survey).

Let $p \in [0,1]$ be a computable real number and μ_p the Bernoulli measure over Cantor space given by $\mu_p[w] = p^{|w|_1}(1-p)^{|w|_0}$. It is well-known[1] that if an infinite binary sequence

[1] see [11] for a generalization of this result

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 567–576
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

x is Martin-Löf random w.r.t. μ_p then $\dim(x) = \mathrm{Dim}(x) = h(\mu_p)$, where $h(\mu_p)$ is the entropy of μ_p defined by

$$h(\mu_p) = -p\log(p) - (1-p)\log(1-p). \tag{1}$$

This result is not difficult to prove and reduces to the strong law of large numbers for Martin-Löf random sequences, as on the one hand[2]

$$K(x\restriction_n) = -\log \mu_p[x\restriction_n] + O(\log(n))$$

for μ_p-random sequences by Levin-Schnorr theorem, and on the other hand

$$-\frac{1}{n}\log \mu_p[x\restriction_n] = -\frac{|x\restriction_n|_1}{n}\log(p) - \frac{|x\restriction_n|_0}{n}\log(1-p)$$

which converge to $h(\mu_p)$ for μ_p-random sequences, by the Strong Law of Large Numbers for Martin-Löf random sequences.

This result highlights the relationship between Shannon's information theory, Kolmogorov algorithmic information theory and effective randomness.

Ergodic theory provides a natural extension of information theory in which many results can be transferred, with more involved proofs, from the case of independent identically distributed random variables to the ergodic case, where independence is only required *asymptotically, in the average* (see Section 2 for a precise definition).

First, the strong law of large numbers extends to Birkhoff's ergodic theorem. Second, the coincidence between local information and entropy extends through the Shannon-McMillan-Breiman theorem. Whether Martin-Löf randomness fits with these theorems has been an open problem for a while. The first results were proved by V'yugin [22], based on non-classical, constructive proofs of the theorems. He proved, in particular:

▶ **Theorem 1** (Effective Birkhoff ergodic theorem I). *Let μ be a computable shift-invariant ergodic measure over $\{0,1\}^{\mathbb{N}}$ and $f \in L^1(\mu)$ be computable. For every Martin-Löf μ-random sequence x,*

$$\lim_{n\to\infty} \frac{1}{n}\sum_{k=0}^{n-1} f \circ T^k(x) = \int f \,\mathrm{d}\mu.$$

The entropy of an ergodic measure is defined as

$$h(\mu) = \lim_{n\to\infty} -\frac{1}{n}\sum_{|w|=n} \mu[w]\log \mu[w]. \tag{2}$$

Observe that (1) and (2) are consistent as they give the same quantity when μ is a Bernoulli measure.

▶ **Theorem 2** (Effective Shannon-McMillan-Breiman theorem I). *Let μ be a computable shift-invariant ergodic measure over $\{0,1\}^{\mathbb{N}}$. For every Martin-Löf μ-random sequence x,*

$$\limsup_{n\to\infty} \frac{K(x\restriction_n)}{n} = \limsup_{n\to\infty} -\frac{1}{n}\log \mu[x\restriction_n] = h(\mu).$$

[2] K is the prefix-free version of Kolmogorov complexity

The question whether $\liminf \frac{K(x\restriction n)}{n}$ coincides with $h(\mu)$ for every Martin-Löf μ-random was left open by V'yugin. An alternative proof of Theorem 2 approximating ergodic measures by Markovian measures was later developed by Nakamura [16], but also left the question open. In this paper we provide a positive answer to this question. As was pointed out by a referee, Hochman's constructive proof of the Shannon-McMillan-Breiman theorem [8] gives another proof of the result.

A classical proof of the Shannon-McMillan-Breiman theorem uses Birkhoff's ergodic theorem, applied to some particular functions. The problem in making it effective is that these functions are not computable in general. Recent works have been achieved to push the effective ergodic theorem to the largest possible class of functions. Here we extend it enough to get the full effective Shannon-McMillan-Breiman theorem.

In Section 2 we recall basic notions of computability, randomness and ergodic theory. In Section 3 we develop effective versions of Birkhoff's ergodic theorem. In Section 4 we present our main result.

2 Background and notations

We work on the Cantor space $\{0,1\}^{\mathbb{N}}$ of infinite binary sequences. A finite word $w \in \{0,1\}^*$ determines the cylinder $[w] \subseteq \{0,1\}^{\mathbb{N}}$ of infinite sequences starting with w. If $x \in \{0,1\}^{\mathbb{N}}$ and $n \in \mathbb{N}$, $x\restriction_n$ is the prefix of x of length n, and is also denoted $x_0x_1\dots x_{n-1}$. The cylinders form a base of the product topology.

Effective topology

An open set $U \subseteq \{0,1\}^{\mathbb{N}}$ is ***effective*** if it is a recursively enumerable union of cylinders. A closed set is effective it its complement is an effective open set. A function $f : \{0,1\}^{\mathbb{N}} \to \mathbb{R}$ is ***computable*** if there is a Turing machine that on oracle x and input n computes a rational number q such that $|q - f(x)| < 2^{-n}$. Equivalently, f is computable if for every rational numbers $a < b$, $f^{-1}(a,b)$ is effectively open, uniformly in a, b. A function $f : \{0,1\}^{\mathbb{N}} \to [0,+\infty]$ is ***lower*** (resp. ***upper***) ***semi-computable*** if there is a Turing machine that on oracle x and input n computes a rational number q_n such that $f(x) = \sup_n q_n$ (resp. $f(x) = \inf_n q_n$). Equivalently, f is lower (resp. upper) semi-computable if for every rational number a, $f^{-1}(a,+\infty]$ (resp. $f^{-1}[0,a)$) is effectively open, uniformly in a.

Kolmogorov complexity and Martin-Löf randomness

For $w \in \{0,1\}^*$, $K(w)$ is the prefix-free version of Kolmogorov complexity, defined by Levin and Chaitin independently. It is defined as the length of a shortest input of a universal Turing machine with prefix-free domain computing w on that input.

A probability measure μ over $\{0,1\}^{\mathbb{N}}$ is determined by its value on the cylinders $\mu[w]$, for $w \in \{0,1\}^*$. μ is computable if all $\mu[w]$ are computable real numbers, uniformly in w. Given a computable probability measure μ, a sequence $x \in \{0,1\}^{\mathbb{N}}$ is ***Martin-Löf μ-random***, denoted $x \in \mathsf{ML}_\mu$, if there is c such that for all n,

$$K(x\restriction_n) \geq -\log \mu[x\restriction_n] - c.$$

Martin-Löf's original definition of a random sequence (in [14]) was expressed in terms of tests rather than complexity, but the one given above, due to Levin and Chaitin [4] independentely, was proved to be equivalent by Levin [9] and Schnorr [20].

Let us briefly present a notion of randomness test that we will use in the sequel. A ***μ-test*** is a lower semi-computable function $f : \{0,1\}^{\mathbb{N}} \to [0,+\infty]$ such that $\int f \, \mathrm{d}\mu \leq 1$. The definition of a Martin-Löf random sequence can be rephrased as follows: $x \in \mathsf{ML}_\mu$ if and only if the quantity

$$d_\mu(x) = \sup_n \{-\log \mu[x\restriction_n] - K(x\restriction_n)\},$$

called the ***randomness deficiency*** of x, is finite. It was proved in [9] that $t_\mu := 2^{d_\mu}$ is a μ-test and by Gács [7] that it is *optimal* in the sense that for every μ-test f, there exists a constant c_f such that $f \leq c_f t_\mu$. As a result, $x \in \mathsf{ML}_\mu$ if and only if $f(x) < \infty$ for each μ-test f if and only if $t_\mu(x) < \infty$. More can be found on this subject in [10] as well as in the recent textbooks [17, 5].

Ergodic theory

We recall some basic notions of ergodic theory, more details can be found in [21, 19]. We denote by $T : \{0,1\}^{\mathbb{N}} \to \{0,1\}^{\mathbb{N}}$ the ***shift map*** defined by $T(x_0x_1x_2\ldots) = x_1x_2x_3\ldots$. A measure μ over $\{0,1\}^{\mathbb{N}}$ is ***shift-invariant*** if for all Borel sets A, $\mu(T^{-1}A) = \mu(A)$, equivalently if $\mu[0w] + \mu[1w] = \mu[w]$ for all $w \in \{0,1\}^*$. μ is ***ergodic*** if for all Borel sets A such that $T^{-1}A = A$ up to a null set, $\mu(A) = 0$ or 1. Equivalently, μ is ergodic if for all $u, v \in \{0,1\}^*$,

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} \mu([u] \cap T^{-k}[v]) = \mu[u] \cdot \mu[v].$$

Every Bernoulli measure is shift-invariant and ergodic.

3 Effective ergodic theorems

The following theorem, taken from [2], extends a result of Kučera from the uniform measure to any ergodic shift-invariant measure:

▶ **Theorem 3** (Effective Poincaré recurrence theorem)**.** *Let μ be a computable ergodic shift-invariant measure and $C \subseteq \{0,1\}^{\mathbb{N}}$ an effective closed set such that $\mu(C) > 0$. Every Martin-Löf μ-random sequence has a tail in C, i.e. for every $x \in \mathsf{ML}_\mu$ there exists k such that $T^k(x) \in C$.*

In [3] and [6] independently this result was used to prove that not only the orbit of x eventually falls into C, but it does so with frequency $\mu(C)$.

▶ **Theorem 4** (Effective Birkhoff ergodic theorem II)**.** *Let μ be a computable ergodic shift-invariant measure and $C \subseteq \{0,1\}^{\mathbb{N}}$ an effective closed set such that $\mu(C) > 0$. For every Martin-Löf μ-random sequence x,*

$$\lim_{n\to\infty} \frac{1}{n} |\{k < n : T^k(x) \in C\}| = \mu(C).$$

We first generalize the result from sets to functions:

▶ **Theorem 5** (Effective Birkhoff ergodic theorem III)**.** *Let μ be a computable ergodic shift-invariant measure. Assume $f : \{0,1\}^{\mathbb{N}} \to [0,+\infty]$ is:*

- *either lower semi-computable,*

- *or upper semi-computable and bounded by a μ-test.*

For each $x \in \mathsf{ML}_\mu$,

$$\lim_{n\to\infty} \sum_{k=0}^{n-1} f \circ T^k(x) = \int f \,\mathrm{d}\mu.$$

Proof. Let us denote the Birkhoff averages by $A_n^f(x) = \frac{1}{n}(f(x) + \ldots + f \circ T^{n-1}(x))$.

If f is lower semi-computable, then there is a sequence of uniformly computable non-negative functions $f_n \nearrow f$. Applying Theorem 1 to f_n and $x \in \mathsf{ML}_\mu$ gives $\liminf_k A_k^f(x) \geq \liminf_k A_k^{f_n}(x) = \int f_n \,\mathrm{d}\mu$. By the monotone convergence theorem, $\int f_n \,\mathrm{d}\mu \nearrow \int f \,\mathrm{d}\mu$, so $\liminf_k A_k^f(x) \geq \int f \,\mathrm{d}\mu$. If $\int f \,\mathrm{d}\mu = \infty$ we are done. Otherwise, let $q > \int f \,\mathrm{d}\mu$ be a rational number. The set $C_K := \{x : \forall k \geq K, A_k^f(x) \leq q\}$ is effectively closed and by the classical ergodic theorem, there exists K such that $\mu(C_K) > 0$. Theorem 3 tells us that if $x \in \mathsf{ML}_\mu$ then there is n such that $T^n(x) \in C_K$. As a result, $\limsup A_k^f(x) = \limsup A_k^f(T^n(x)) \leq q$. As this is true for every $q > \int f \,\mathrm{d}\mu$, we get the result.

Now, if f is upper semi-computable and $f \leq t$ where t is a μ-test, then for $x \in \mathsf{ML}_\mu$, applying the preceding result to t and $t - f$,

$$A_n^f(x) = A_n^t(x) - A_n^{(t-f)}(x) \to \int t \,\mathrm{d}\mu - \int (t-f) \,\mathrm{d}\mu = \int f \,\mathrm{d}\mu. \qquad \blacktriangleleft$$

We then extend this result further:

▶ **Corollary 6** (Effective Birkhoff ergodic theorem IV). *Let $f : \{0,1\}^{\mathbb{N}} \to [0, +\infty]$ be Δ_2^0 on ML_μ, i.e. there is a sequence f_n of uniformly computable functions such that $f(x) = \lim_n f_n(x)$ for each $x \in \mathsf{ML}_\mu$. Assume that f is dominated by a μ-test. For every $x \in \mathsf{ML}_\mu$,*

$$\lim_{n\to\infty} \sum_{k=0}^{n-1} f \circ T^k(x) = \int f \,\mathrm{d}\mu.$$

Proof. Let $g_N = \inf_{n \geq N} f_n$ and $h_N = \min(t, \sup_{n \geq N} f_n)$ where t is a μ-test dominating f. On ML_μ, $g_N \nearrow f$ and $h_N \searrow f$. By the monotone and dominated convergence theorem, the convergences hold in $L^1(\mu)$. Applying Theorem 4 to g_N and h_N gives the result. More precisely, for every $x \in \mathsf{ML}_\mu$ and every N,

$$\liminf_n A_n^f(x) \geq \liminf_n A_n^{g_N}(x) = \int g_N \,\mathrm{d}\mu$$

$$\limsup_n A_n^f(x) \leq \limsup_n A_n^{h_N}(x) = \int h_N \,\mathrm{d}\mu,$$

so

$$\int f \,\mathrm{d}\mu = \sup_N \int g_N \,\mathrm{d}\mu \leq \liminf_n A_n^f(x) \leq \limsup_n A_n^f(x) \leq \inf_N \int h_N \,\mathrm{d}\mu = \int f \,\mathrm{d}\mu. \qquad \blacktriangleleft$$

3.1 Further results

We briefly discuss the extent to which the assumptions in the preceding results are needed.

3.1.1 Δ_2^0 functions

In Corollary 6 that one cannot get rid of the assumption that f is dominated by a μ-test, as a limit of uniformly computable functions may not be finite on all Martin-Löf random

points, even if it is integrable. Let us give an example of such an f. Let μ be the uniform measure over $[0,1]$ and α be a Δ^0_2 random real (a Δ^0_2 real is a limit of a sequence of uniformly computable reals). Define f by $f(x) = \frac{1}{\sqrt{|x-\alpha|}}$ for $x \neq \alpha$ and $f(\alpha) = +\infty$. f is a limit of uniformly computable functions. Indeed, let a_n be a computable sequence of reals converging to α: for all x, $f(x) = \lim_{n\to\infty} \frac{1}{2^{-n}+\sqrt{|x-a_n|}}$.

3.1.2 Upper semi-computable functions

In Theorem 5 we do not know whether the assumption that the upper semi-computable function f is dominated by a test is really needed. Without this assumption, we still get a partial result.

▶ **Proposition 7.** *Let $f : \{0,1\}^{\mathbb{N}} \to [0,+\infty]$ be upper semi-computable. For every $x \in \mathsf{ML}_\mu$,*

$$\liminf_{n\to\infty} \sum_{k=0}^{n-1} f \circ T^k(x) = \int f \,\mathrm{d}\mu.$$

Proof. The argument to prove inequality $\geq$ is essentially the one used in Theorem 5 to prove that $\limsup A^f_n(x) \leq \int f \,\mathrm{d}\mu$ when f is lower semi-computable. Let $q < \int f \,\mathrm{d}\mu$ be a rational number. By the classical ergodic theorem there exists $K \in \mathbb{N}$ such that the effective closed set $C_{q,K} := \{x : \forall k \geq K, A^f_k(x) \geq q\}$ has positive measure. By Theorem 3, if $x \in \mathsf{ML}_\mu$ then there exists n such that $T^n(x) \in C_{q,K}$, which implies $\liminf A^f_k(x) = \liminf A^f_k(T^n(x)) \geq q$. Taking q closer and closer to $\int f \,\mathrm{d}\mu$ we get $\liminf A^f_k(x) \geq \int f \,\mathrm{d}\mu$.

We now prove the other inequality. If $\int f \,\mathrm{d}\mu = +\infty$ we are done, otherwise let $q > \int f \,\mathrm{d}\mu$ be a rational number. By the classical ergodic theorem, for each $K \in \mathbb{N}$, $\mu(C_{q,K}) = 0$ so $C_{q,K}$ contains no μ-random point. As a result, $\liminf A^f_k(x) \leq \int f \,\mathrm{d}\mu$ for each $x \in \mathsf{ML}_\mu$. ◀

Observe that it is possible to build an integrable upper semi-computable function f that is not dominated by a μ-test.

3.1.3 Π^0_2 sets

By Corollary 6, if D is a Δ^0_2-set, i.e. if both D and its complement are effective intersections of effective open sets, then the visit frequency of the trajectory of a Martin-Löf random sequence into D is always $\mu(D)$. What can be said about sets of higher complexity in the effective Borel hierarchy?

Let Ω be a left-c.e. random sequence (a Chaitin's Ω). Let $D_n = T^n[\Omega, \Omega + 2^{-2n-2}]$. D_n is Σ^0_2 (and even Δ^0_2), uniformly in n: $D_n = T^n(\bigcup_i [\Omega, 1] \cap [0, q_i + 2^{-2n-2}])$ where $q_i \nearrow \Omega$ is a computable sequence of dyadic numbers. One easily checks that $\mu(D_n) \leq 2^{-n-2}$. Let $D = \bigcup_n D_n$: D is Σ^0_2, $\mu(D) < 1$ but all the iterates of Ω belong to D.

As a result, the complement of D is a Π^0_2-set of positive measure, but no iterate of Ω belongs to this set. We can conclude that the complement of D does not contain any effective closed set of positive measure. As shown by the following result, the converse is also true: if a Π^0_2-set of positive measure contains no effective closed set of positive measure then there is a random sequence whose iterates avoid this set.

▶ **Proposition 8.** *Let $D = \bigcap_n U_n$ where U_n are (not necessarily uniformly) effective open sets. Assume $\mu(D) > 0$. The following are equivalent:*

1. *every random point eventually falls into D,*
2. *D contains an effective closed set of positive measure,*
3. *$\liminf A^{\mathbf{1}_D}_n(x) > 0$ for every random point x.*

Proof. $1 \Rightarrow 2$. Let K_0 be the complement of some level in a universal ML test.

Assume D contains no effective closed set of positive measure. We construct a decreasing sequence K_n of effective closed sets of positive measure. As $T^n(K_n)$ has positive measure, it is not a subset of D so there is i_{n+1} such that $T^n(K_n) \setminus U_{i_{n+1}} \neq \emptyset$. Let $K_{n+1} = K_n \setminus T^{-n}U_{i_{n+1}}$: $K_{n+1} \neq \emptyset$ so $\mu(K_{n+1}) > 0$ as it is an effective closed set that contains random sequences. In the limit, $\bigcap_n K_n$ is non-empty and if $x \in \bigcap_n K_n$ then for every n, $T^n(x) \notin U_{i_{n+1}}$, so $T^n(x) \notin D$.

$2 \Rightarrow 3$. Direct: if $C \subseteq D$ is an effective closed set of positive measure then $\liminf A_n^{\mathbf{1}_D} \geq \liminf A_n^{\mathbf{1}_C} = \mu(C) > 0$ by Theorem 3

$3 \Rightarrow 1$. Obvious. ◀

3.2 Recurrence time

Let (X, T) be a dynamical system. Given a set A and a point $x \in A$, the recurrence time of x is defined as the minimal $k \geq 1$ such that $T^k(x)$ belongs to A. Ergodic theory provides several results about the asymptotic behavior of the recurrence time of a point, which have applications in coding and information theory. We focus on the particular case of the shift map on the Cantor space and when A is a cylinder.

For $x \in \{0,1\}^{\mathbb{N}}$, let $R_n(x) := \min\{k \geq n : x_0 \dots x_{n-1} = x_k \dots x_{x+n-1}\} = \min\{k \in n : x\restriction_n = T^k(x)\restriction_n\}$. Ornstein and Weiss [18] proved that for a shift-invariant ergodic probability measure μ, $\log R_n(x)/n$ converge to the entropy $h(\mu)$ almost-surely, extending the convergence in probability earlier proved by Wyner and Ziv [23]. Nakamura [16] proved a weak version of that result for Martin-Löf random points, showing that $\limsup \log R_n(x)/n = h(\mu)$ for every Martin-Löf μ-random sequence x.

Here we show that the full result can be simply derived from Theorem 3.

▶ **Theorem 9.** *Let μ be a shift-invariant ergodic measure. For every $x \in \mathsf{ML}_\mu$,*

$$\lim_{n\to\infty} \frac{\log R_n(x)}{n} = h(\mu).$$

Proof. Let $f_n(x) = \log R_n(x)/n$ and $q < h(\mu)$ be rational. Note that f_n is computable on ML_μ, uniformly in n. From Ornstein and Weiss result, the set $\{x : \exists N \forall n \geq N, f_n(x) \geq q\}$ has measure one, hence there exists N such that the set $C_N := \{x : \forall n \geq N, f_n(x) \geq q\}$ has positive measure. Let $x \in \mathsf{ML}_\mu$: as C_N is effectively closed, by Theorem 3 there exists k such that $T^k(x) \in C_N$, which implies $\liminf f_n(T^k(x)) \geq q$. One easily sees that $R_n(x) \geq R_{n-1}(T(x))$, which implies $\liminf f_n(x) \geq \liminf f_n(T^k(x)) \geq q$. As this inequality holds for each $q < h(\mu)$, we get $\liminf f_n(x) = h(\mu)$ for every $x \in \mathsf{ML}_\mu$. Together with Nakamura's result, it proves the theorem. ◀

4 The effective Shannon-McMillan-Breiman theorem

We now present our main result.

▶ **Theorem 10** (Effective Shannon-McMillan-Breiman theorem II). *Let μ be a computable shift-invariant probability measure. For each $x \in \mathsf{ML}_\mu$,*

$$\lim_{n\to\infty} \frac{K(x\restriction_n)}{n} = \lim_{n\to\infty} -\frac{1}{n} \log \mu[x\restriction_n] = h(\mu).$$

A proof of the classical result, stating the result for a.e. x, can be found in [21, 19]. It makes use of martingale convergence theorems and ergodic theorems. The main difficulty in

adapting the proof is to make sure that the effective versions of the ergodic theorem can be applied. The rest of this section is devoted to the proof of Theorem 10.

An easy calculation shows that

$$-\log \mu[x\restriction_n] = \sum_{k=0}^{n-1} f_{n-1-k} \circ T^k(x) \tag{3}$$

where

$$f_k(x) := -\log \mu[x_0|x_1 \dots x_k] = -\log \frac{\mu[x_0 \dots x_k]}{\mu[x_1 \dots x_k]} \quad \text{for } k \geq 1,$$

$$f_0(x) := -\log \mu[x_0].$$

▶ **Lemma 11.** *$f_k(x)$ converge for each $x \in \mathsf{ML}_\mu$.*

Proof. Define the computable martingale

$$\begin{aligned} d(\epsilon) &= 2 \\ d(x_0) &= \frac{1}{\mu[x_0]} \\ d(x_0 \dots x_k) &= \frac{\mu[x_1 \dots x_k]}{\mu[x_0 \dots x_k]} \quad \text{for } k \geq 1. \end{aligned}$$

By the effective Doob's convergence theorem (see Theorem 7.1.3 on page 270 in [5]), for each $x \in \mathsf{ML}_\mu$, $d(x_0 \dots x_k)$ converges, and so does $f_k(x) = \log d(x_0 \dots x_k)$. ◀

Let $f(x)$ be the limit. We write

$$-\frac{1}{n}\log \mu[x\restriction_n] = \frac{1}{n}\sum_{k=0}^{n-1}(f_{n-1-k} \circ T^k(x) - f \circ T^k(x)) + \frac{1}{n}\sum_{k=0}^{n-1} f \circ T^k(x)$$

and prove that the first term tends to 0 while the second term converges to $\int f \,\mathrm{d}\mu = h(\mu)$.

We will use the following lemma (Corollary 2.2 on page 261 in [19], Lemma 4.26 on page 26 in [21]).

▶ **Lemma 12.** *$f^* := \sup_k f_k \in L^1$.*

As $f_k \to f$ a.e. and the convergence is dominated by $f^* \in L^1$, $f_k \to f$ in L^1.

▶ **Proposition 13.** *For each $x \in \mathsf{ML}_\mu$,*

$$\lim_n \frac{1}{n}\sum_{k=0}^{n-1} f \circ T^k(x) = \int f \,\mathrm{d}\mu = h_\mu(P). \tag{4}$$

Proof. That $\int f \,\mathrm{d}\mu = h(\mu)$ is a classical result and follows from $h(\mu) = \lim_k \int f_k \,\mathrm{d}\mu$ and the L^1-convergence of f_k to f.

f^* is lower semi-computable and by Lemma 12 it is a μ-test. By construction, f is Δ^0_2 on ML_μ and it is dominated by f^* so it satisfies the conditions of Corollary 6, from which the result follows directly. ◀

▶ **Proposition 14.** *For each $x \in \mathsf{ML}_\mu$,*

$$\lim_{n\to\infty} \frac{1}{n}\sum_{k=0}^{n-1} f_{n-1-k} \circ T^k(x) - f \circ T^k(x) = 0. \tag{5}$$

Proof. Let

$$g_N = \sup_{k\geq N} |f_k - f| \quad \text{and} \quad \tilde{g}_N = \sup_{k,j\geq N} |f_k - f_j|.$$

For $x \in \mathsf{ML}_\mu$,

$$\begin{aligned}|f_k(x) - f(x)| &= \lim_j |f_k(x) - f_j(x)| \\ &= \limsup_j |f_k(x) - f_j(x)| \\ &\leq \sup_{j\geq N} |f_k(x) - f_j(x)|,\end{aligned}$$

so $g_N(x) \leq \tilde{g}_N(x)$. As $f_k \to f$ a.e., $\tilde{g}_N \to 0$ a.e. As $\tilde{g}_N \leq 2f^* \in L^1$, $\tilde{g}_N \to 0$ in L^1 by the dominated convergence theorem. On ML_μ,

$$\begin{aligned}\left|\frac{1}{n}\sum_{k=0}^{n-1} f_{n-1-k}\circ T^k - f\circ T^k\right| &\leq \frac{1}{n}\sum_{k=0}^{n-1} |f_{n-1-k}\circ T^k - f\circ T^k| \\ &= \frac{1}{n}\sum_{k=0}^{n-1-N} |f_{n-1-k}\circ T^k - f\circ T^k| + \frac{1}{n}\sum_{k=n-N}^{n-1} |f_{n-1-k}\circ T^k - f\circ T^k| \\ &\leq \frac{1}{n}\sum_{k=0}^{n-1-N} g_N\circ T^k + \frac{1}{n}\sum_{k=n-N}^{n-1} (f^*+f)\circ T^k \\ &\leq \frac{1}{n}\sum_{k=0}^{n-1-N} \tilde{g}_N\circ T^k + \frac{1}{n}\sum_{k=0}^{n-1}(f^*+f)\circ T^k - \frac{1}{n}\sum_{k=0}^{n-N-1}(f^*+f)\circ T^k.\end{aligned}$$

Fix N and let $n \to \infty$. As $\tilde{g}_N \in L^1$ is lower semi-computable, the first term converges to $\int \tilde{g}_N \,\mathrm{d}\mu$ by the Effective Ergodic Theorem 5. As $f^* + f$ is Δ^0_2 on ML_μ and is dominated by the μ-test $2f^*$, the second and the third terms converge to $\int (f^* + f)\,\mathrm{d}\mu$ by Corollary 6 so their limits cancel each other.

As $\int \tilde{g}_N \,\mathrm{d}\mu \to 0$, we have proved equality (5). ◀

Putting equalities (3), (4) and (5) together gives, for $x \in \mathsf{ML}_\mu$,

$$\lim_n -\frac{1}{n}\log \mu[x\restriction_n] = \lim_n \frac{1}{n}\sum_{k=0}^{n-1} f_{n-1-k}\circ T^k(x) = h(\mu).$$

Acknowledgments

The author would like to thank the anonymous referees for their remarks and for pointing out Hochman's results.

References

1 Krishna B. Athreya, John M. Hitchcock, Jack H. Lutz, and Elvira Mayordomo. Effective strong dimension in algorithmic information and computational complexity. *SIAM J. Comput.*, 37(3):671–705, 2007.

2 Laurent Bienvenu, Adam Day, Ilya Mezhirov, and Alexander Shen. Ergodic-type characterizations of algorithmic randomness. In *Computability in Europe (CIE 2010)*, volume 6158 of *Lecture Notes in Computer Science*, pages 49–58. Springer, 2010.

3 Laurent Bienvenu, Adam R. Day, Mathieu Hoyrup, Ilya Mezhirov, and Alexander Shen. A constructive version of Birkhoff's ergodic theorem for Martin-Löf random points. To appear in *Information and Computation*. ArXiv 1007.5249, 2010.

4 Gregory J. Chaitin. A theory of program size formally identical to information theory. *J. ACM*, 22(3):329–340, 1975.

5 Rod Downey and Denis Hirschfeldt. *Algorithmic Randomness and Complexity*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2010.

6 Johanna N.Y. Franklin, Noam Greenberg, Joseph S. Miller, and Keng Meng Ng. Martin-Löf random points satisfy Birkhoff's ergodic theorem for effectively closed sets. To appear in the *Proceedings of the American Mathematical Society*, 2010.

7 Péter Gács. Exact expressions for some randomness tests. *Z. Math. Log. Grdl. M.*, 26:385–394, 1980.

8 Michael Hochman. Upcrossing inequalities for stationary sequences and applications. *The Annals of Probability*, 37(6):2135–2149, 2009.

9 Leonid A. Levin. On the notion of a random sequence. *Soviet Mathematics Doklady*, 14:1413–1416, 1973.

10 Ming Li and Paul M. B. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Berlin, 1993.

11 Jack Lutz. Gales and the constructive dimension of individual sequences. In Ugo Montanari, José Rolim, and Emo Welzl, editors, *Automata, Languages and Programming*, volume 1853 of *Lecture Notes in Computer Science*, pages 902–913. Springer Berlin / Heidelberg, 2000.

12 Jack H. Lutz. Dimension in complexity classes. In *IEEE Conference on Computational Complexity*, pages 158–169, 2000.

13 Jack H. Lutz. Effective fractal dimensions. *Mathematical Logic Quarterly*, 51(1):62–72, 2005.

14 Per Martin-Löf. The definition of random sequences. *Information and Control*, 9(6):602–619, 1966.

15 Elvira Mayordomo. A Kolmogorov complexity characterization of constructive Hausdorff dimension. *Inf. Process. Lett.*, 84(1):1–3, 2002.

16 Masahiro Nakamura. Ergodic theorems for algorithmically random sequences. *Proceedings of the Symposium on Information Theory and Its Applications*, 28:71–74, 2005.

17 A. Nies. *Computability and randomness*. Oxford logic guides. Oxford University Press, 2009.

18 Donald S. Ornstein and Benjamin Weiss. Entropy and data compression schemes. *IEEE Transactions on Information Theory*, 39:78–83, 1993.

19 Karl Petersen. *Ergodic Theory*. Cambridge Univ. Press, 1983.

20 Claus-Peter Schnorr. Process complexity and effective random tests. *J. Comput. Syst. Sci.*, 7(4):376–388, 1973.

21 Meir Smorodinsky. *Ergodic Theory, Entropy*, volume 214 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin Heibelberg New York, 1971.

22 Vladimir V. V'yugin. Ergodic theorems for individual random sequences. *Theoretical Computer Science*, 207(4):343–361, 1998.

23 Aaron D. Wyner and Jacob Ziv. Some asymptotic properties of the entropy of a stationary ergodic data source with applications to data compression. *IEEE Transactions on Information Theory*, 35:1250–1258, 1989.

The Field of Reals is not ω-Automatic

Faried Abu Zaid[1], Erich Grädel[1], and Łukasz Kaiser[2]

1 **Mathematische Grundlagen der Informatik, RWTH Aachen University**
{abuzaid,graedel}@logic.rwth-aachen.de
2 **CNRS & LIAFA, Université Paris Diderot – Paris 7**
kaiser@liafa.jussieu.fr

Abstract

We investigate structural properties of ω-automatic presentations of infinite structures in order to sharpen our methods to determine whether a given structure is ω-automatic. We apply these methods to show that no field of characteristic 0 admits an injective ω-automatic presentation, and that uncountable fields with a definable linear order cannot be ω-automatic.

1998 ACM Subject Classification F.4.1 Mathematical Logic

Keywords and phrases Logic, Algorithmic Model Theory, Automatic Structures

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.577

1 Introduction

Automatic structures are (in general) infinite structures that admit a finite presentation by automata. Informally, an automatic presentation of a relational structure $\mathfrak{B} = (B, R_1, \ldots, R_m)$ consists of a language L, which must be recognisable by an automaton $\mathcal{A}$, and a surjective function $\pi : L \to B$ that associates every word of L with the element of B that it represents. The function π must be surjective (every element of B is named by some word in L) but need not be injective (elements may have more than one name). In addition it must be recognisable by automata, reading their inputs synchronously, whether two elements of L name the same element of B, and, for each relation R_i, whether a given tuple of words in L names a tuple in R_i. Together, the automaton $\mathcal{A}$ and the automata that recognise equality and the relations $R_1, \ldots, R_m$ provide a finite representation of the structure $\mathfrak{B}$. This concept has implicitly been known since the first days of automata theory, but the systematic investigation of such presentations was mainly invoked by the work of Khoussainov and Nerode [7] and later by the work of Blumensath and Grädel [1].

In principle we can use automata over finite words, infinite words, finite trees, or infinite trees to obtain different classes of automatic structures. Indeed, any model of automata can be used for this approach as long as it is effectively closed under all first-order operations (union, intersection, complementation, and projection) and its emptiness problem is decidable.

As a consequence of these two properties it follows that

- every automatic structure has a decidable first-order theory and, more generally,
- given any automatic presentation of $\mathfrak{A}$ and any first-order formula $\varphi(x_1, \ldots, x_k)$ one can effectively construct an automaton that represents the relation $\varphi^{\mathfrak{A}} = \{\bar{a} \in A^k : \mathfrak{A} \models \varphi(\bar{a})\}$.

Thus, all definable properties of automatic structures can be algorithmically investigated using automata-theoretic methods based on appropriate finite presentations. This makes automatic structures a domain of considerable interest for computer science.

While the case of word-automatic structures (with presentations based on automata operating on finite words) is reasonably well understood, much less is known for presentations

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 577–588

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


based on other classes of automata. In particular, our methods to establish whether a given structure admits an automatic presentation on, say, infinite words, or to classify all such structures inside a given domain, are still relatively weak. For countable structures, an important achievement is the result by Kaiser, Rubin, and Bárány [6] that a countable structure is ω-automatic if, and only if, it is word-automatic. Thus the recent result by Tsankov [14] that the additive group of the rationals is not automatic immediately implies that it is not ω-automatic either. Another interesting result for word-automatic structures is due to Khoussainov, Nies, Rubin and Stephan. In [8] among others they show that there are no infinite automatic fields.

One of the most prominent and important structures with a decidable first-order theory is certainly the field of reals $(\mathbb{R}, +, \cdot)$. The decidability goes back to Tarski [13] and is based on a quantifier elimination argument. Therefore, it is very natural to ask whether the field of reals admits an automatic presentation. Of course, such a presentation cannot be based on automata on finite words (or finite trees) because languages of finite words and trees are countable. However, it might be the case that the field of reals is ω-automatic, i.e., admits a presentation based on automata on infinite words, or that it is ω-tree-automatic, with a presentation based on automata on infinite trees.

The question whether this is the case is closely related to classical problems raised by Büchi and Rabin in the context of decidable first-order theories. The decidability of Presburger arithmetic, the first-order theory of $(\mathbb{N}, +)$, had originally been proved by quantifier elimination, but Büchi's automata-based proof of the decidability of S1S (the monadic theory of infinite words) easily carries over to an automata-theoretic decidability argument for Presburger arithmetic. And indeed, $(\mathbb{N}, +)$ is now one of the standard examples for word-automatic structures. In Rabin's classical paper [11], where he proved the decidability of S2S (the monadic theory of the infinite binary tree) and several other theories, he explicitly raised the question whether also the decidability of the field of reals could be proved by automata-theoretic methods.

In this paper we investigate the question whether the field of reals, and other fields of characteristic 0, are ω-automatic. It is quite easy to see that both reducts $(\mathbb{R}, +)$ and $(\mathbb{R}, \cdot)$ of the field of reals are ω-automatic but it has so far been open whether two such presentations could be combined to one of the entire field. We shall prove that this is not the case. More generally, we establish the following results.

▶ **Theorem 1.** *No field of characteristic 0 admits an injective ω-automatic presentation.*

▶ **Theorem 2.** *No field of characteristic 0 with a definable linear order is ω-automatic.*

In particular, the field of reals does not admit an ω-automatic presentation. Notice that this does not completely settle Rabin's question, since it might still be the case that the field of reals is ω-tree-automatic. We do not expect this, but our current methods do not seem to be powerful enough to settle this question.

We briefly outline our methodology. We shall make heavy use of the notion of *end-equivalence* of infinite words: two words $v, w \in \Sigma^\omega$ are end-equivalent, in short $v \sim_e w$, if they are equal from some position onwards, i.e. $v[n, \omega) = w[n, \omega)$ for some $n \in \mathbb{N}$, and we analyse the size of $\sim_e$-equivalent sets in ω-automatic structures.

In the study of ω-automatic presentations, it is important to distinguish between injective and non-injective presentations. Injective presentations are much easier to work with, and, of course, they have the advantage that we do not need an automaton to determine whether two words encode the same elements. In the case of word-automatic structures it is not

hard to see that we can always find an injective automatic presentation, which makes our life much easier. On the other side, it had been open for some time whether ω-automatic structures always admit injective presentations, until Hjorth, Khoussainov, Montalbán and Nies [5] were able to describe an ω-automatic structure that does not even permit an injective Borel presentation (which is a much more general notion than an injective ω-automatic presentation).

Nevertheless, many ω-automatic structures do admit injective presentations such as, for instance, the reducts $(\mathbb{R}, +)$ and $(\mathbb{R}, \cdot)$ of the field of reals. Therefore it is also interesting to ask what kind of structures admit an injective presentation.

Our first step will be the observation that every injective ω-automatic presentation of an infinite structure necessarily produces an infinite set of $\sim_e$-equivalent elements. On the other side, we shall prove the following general fact on ω-automatic fields.

▶ **Lemma 3.** *Whenever a field of characteristic 0 admits an ω-automatic presentation then the size of all $\sim_e$-equivalent subsets of the field is bounded by a constant.*

To establish this, we shall combine an argument saying that the image of every set of $\sim_e$-equivalent elements under a definable k-ary function can be partitioned into a small number of sets, each of which consists of $\sim_e$-equivalent elements only, with a result on additive combinatorics, known as Freiman's Theorem.

From these results, we immediately obtain Theorem 1 because on one side, an injective presentation of an infinite filed would have to contain an infinite set of $\sim_e$ equivalent elements, but on the other side, we have proved that every such set should be of bounded size.

To extend this result beyond injective presentations, more work is needed. We shall assume that our fields admit a definable linear order, and prove a strengthening of Kuske's result [9] that $(\{0,1\}^\omega, <_{lex})$ is embeddable into every ω-automatic uncountable linear order. In fact, we shall prove that every ω-automatic presentation of an uncountable linear order contains an injective automatic presentation of $(\{0,1\}^\omega, <_{lex})$. This implies that every ω-automatic presentation of an uncountable structure with a definable linear order contains an infinite $\sim_e$-equivalent set. Together with Lemma 3 this implies Theorem 2.

2 Preliminaries

First, we give a formal definition of an ω-automatic presentation over the alphabet Σ.

▶ **Definition 4.** An *ω-automatic presentation* of a structure $\mathfrak{A} = (A, R_1^{\mathfrak{A}}, \ldots, R_n^{\mathfrak{A}})$ consists of a structure $\mathcal{L} = (L, \approx, R_1^{\mathcal{L}}, \ldots, R_n^{\mathcal{L}})$, $L \subseteq \Sigma^\omega$, and a surjective function $\pi : L \to A$ such that:

- $\approx = \{(v, w) : \pi(v) = \pi(w)\}$,
- $R_i^{\mathcal{L}} = \{(v_1, \ldots, v_{r_i}) : (\pi(v_i), \ldots, \pi(v_{r_i})) \in R_i\}$ for $i = 1, \ldots, n$,
- $L, \approx, R_1^{\mathcal{L}}, \ldots, R_n^{\mathcal{L}}$ are all ω-regular.

We call a presentation injective if π is injective. In this case we will omit $\approx$ in the signature of $\mathcal{L}$. A structure is ω-automatic if it has an ω-automatic presentation.

Sometimes we are also not interested in the concrete labelling function π. In these cases we will also omit π and just demand that that $(L, R_1, \ldots, R_n)/\approx$ is isomorphic to $\mathfrak{A}$.

A relation $R \subseteq L^n$ is called ω-regular if the language

$$L_R = \{\langle v_1, \ldots, v_n\rangle \ : \ (v_1, \ldots, v_n) \in R\} \subseteq (\Sigma^n)^\omega$$

is ω-regular, where $\langle v_1, \ldots, v_n\rangle \in (\Sigma^n)^\omega$ is the so called convolution, defined by

$$\langle v_1, \ldots, v_n\rangle[i] = (v_1[i], \ldots, v_n[i]).$$

2.1 ω-Semigroups

The fundamental correspondence between recognisability by finite automata and by finite semigroups has been extended to ω-regular sets. This is based on the notion of *ω-semigroups*. Rudimentary facts on ω-semigroups are well presented in [10], we only mention what is most necessary.

An ω-semigroup $S = (S_f, S_\omega, \cdot, *, \pi)$ is a two-sorted algebra, where $(S_f, \cdot)$ is a semigroup, $* : S_f \times S_\omega \mapsto S_\omega$ is the *mixed product* satisfying, for every $s, t \in S_f$ and every $\alpha \in S_\omega$, the equality $s \cdot (t * \alpha) = (s \cdot t) * \alpha$, and where $\pi : S_f^\omega \mapsto S_\omega$ is the *infinite product* satisfying $s_0 \cdot \pi(s_1, s_2, \ldots) = \pi(s_0, s_1, s_2, \ldots)$ as well as the associativity rule:

$$\pi(s_0, s_1, s_2, \ldots) = \pi(s_0 s_1 \cdots s_{k_1}, s_{k_1+1} s_{k_1+2} \cdots s_{k_2}, \ldots)$$

for every sequence $(s_i)_{i \geq 0}$ of elements of S_f and every strictly increasing sequence $(k_i)_{i \geq 0}$ of indices. For $s \in S_f$ we write s^ω for $\pi(s, s, \ldots)$.

Morphisms of ω-semigroups are defined to preserve all three products as expected. There is a natural way to extend finite semigroups and their morphisms to ω-semigroups. As in semigroup theory, idempotents play a central role in this extension. An *idempotent* is a semigroup element $e \in S$ satisfying $ee = e$. For every element s in a finite semigroup, the sub-semigroup generated by s contains a unique idempotent s^k. The least $k > 0$ such that s^k is idempotent for every $s \in S_f$ is called the *exponent* of the semigroup S_f. Another useful notion is absorption: we say that s *absorbs* t (on the right) if $st = s$.

There is also a natural extension of the free semigroup Σ^+ to the ω-semigroup $(\Sigma^+, \Sigma^\omega)$ with $*$ and π determined by concatenation. An ω-semigroup $S = (S_f, S_\omega)$ *recognises* a language $L \subseteq \Sigma^\omega$ via a morphism $\phi : (\Sigma^+, \Sigma^\omega) \to (S_f, S_\omega)$ if $\phi^{-1}(\phi(L)) = L$. This notion of recognisability coincides with that by non-deterministic Büchi automata; constructions from Büchi automata to ω-semigroups and back are also presented in [10].

▶ **Theorem 5** ([10]).
A language $L \subseteq \Sigma^\omega$ is ω-regular if, and only if, it is recognised by a finite ω-semigroup.

3 Freiman's Theorem

On the algebraic side, we will use a result on additive combinatorics due to Freiman. It has also been an essential ingredient in Tsankov's recent proof that the additive group of rationals is not automatic [14]. Freiman's Theorem is concerned with subsets A of a group G with small doubling $A + A = \{a + a' : a, a' \in A\}$, in the sense that $|A + A| \leq c|A|$ for some constant $c > 0$. For example, consider the case $G = (\mathbb{Z}, +)$. Then, for every arithmetic progression A, we have $|A + A| < 2|A|$. For larger values of c, more complicated sets than simple arithmetic progressions are possible. To state Freiman's Theorem we need the notion of a *generalised* or *multidimensional arithmetic progression*.

▶ **Definition 6.** A d-dimensional arithmetic progression is a set P such that there exist $p_0 \in P$, $\overline{p} = (p_1, \ldots, p_d) \in P^d$ and $n_1, \ldots, n_d \in \mathbb{N}$ with

$$P = \left\{ p_0 + \sum_{i=1}^{d} m_i p_i \ : \ 0 \leq m_i \leq n_i \text{ for } i = 1, \ldots, d \right\}.$$

A progression is called proper if for some such $p_0, \overline{p}, \overline{n}$ it holds that $|P| = \prod_{1 \leq i \leq d}(n_i + 1)$.

Freiman's Theorem [3] states that every set A with $|A + A| \leq c|A|$ is contained in a proper multidimensional progression P which is not much larger than A.

▶ **Theorem 7** (Freiman). *Let $(G,+)$ be a torsion free Abelian group. For every constant $c \in \mathbb{N}$, there exist $d,k \in \mathbb{N}$ such that every finite subset $A \subseteq G$ with $|A+A| \leq c|A|$ is contained in some proper d-dimensional arithmetic progression P of size at most $k|A|$.*

For proofs we refer to [12] or to the very nice lecture notes [4]. In our analysis of ω-automatic fields, we will need a consequence of Freiman's Theorem which ensures that a fairly large portion of such a set A is contained in an (one-dimensional) arithmetic progression.

▶ **Lemma 8.** *Let $(G,+)$ be a torsion free Abelian group. For every constant c there exist d,k such that for every finite subset $A \subseteq G$ with $|A+A| \leq c|A|$ there exists an arithmetic progression P with $|P \cap A| \geq \sqrt[d]{|A|}/k$.*

Proof. Since $|A+A| \leq c \cdot |A|$, by Freiman's Theorem there is a proper progression $Q = \left\{ q_0 + \sum_{i=1}^{d} m_i q_i \,:\, 0 \leq m_i \leq n_i \text{ for } i = 1,\ldots,d \right\}$ such that $A \subseteq Q$ and $|Q| \leq k \cdot |A|$. We may assume that $n_d \geq n_i$ for all $i < d$, which implies $n_d + 1 \geq \sqrt[d]{|A|}$.

We will now take a closer look at the representation of A in Q. For every tuple $\overline{m} = (m_1,\ldots,m_{d-1}) \leq (n_1,\ldots,n_{d-1})$ we write $A_{\overline{m}}$ for the set

$$A_{\overline{m}} := \left\{ q_0 + \sum_{i=1}^{d-1} m_i q_i + m q_d \,:\, m \leq n_d \right\} \cap A.$$

Clearly, every set $A_{\overline{m}}$ is contained in the induced one dimensional arithmetic progression $P_{\overline{m}} = \{p_0 + n p_1 \,:\, n \leq n_d\}$ with $p_0 = q_0 + \sum_{i=1}^{d-1} m_i q_i$ and $p_1 = q_d$.

All we need to show now is that there is an $\overline{m}$ with $|A_{\overline{m}}| \geq \sqrt[d]{|A|}/k$. Since Q is proper, it holds that $|Q| = \prod_{i=1}^{d}(n_i+1)$. The sets $A_{\overline{m}}$ with $m_i \leq n_i$ form a partition of A into $\prod_{i=1}^{d-1}(n_i+1)$ sets. It follows that there is a tuple $\overline{m}$ with

$$|A_{\overline{m}}| \geq \frac{|A|}{\prod_{i=1}^{d-1}(n_i+1)} \geq \frac{|Q|/k}{\prod_{i=1}^{d-1}(n_i+1)} = \frac{\prod_{i=1}^{d}(n_i+1)}{k\prod_{i=1}^{d-1}(n_i+1)} = \frac{n_d+1}{k} \geq \frac{\sqrt[d]{|A|}}{k}.$$

◀

4 End-Equivalence

Two words $v,w \in \Sigma^\omega$ are end-equivalent, in short $v \sim_e w$, if they are equal from some position onwards, i.e. $v[n,\omega) = w[n,\omega)$ for some $n \in \mathbb{N}$. Making explicit a position m after which the words are equal, we obtain refined relations $\sim_e^m$; two words are $\sim_e^m$-equivalent (m-end-equivalent) if $v[m,\omega) = w[m,\omega)$. Clearly $v \sim_e w$ if, and only if, $v \sim_e^m w$ for some m and the $\sim_e^m$-classes partition any language into finite classes of size at most $|\Sigma|^m$.

End-equivalence plays an important role in the study of ω-regular languages. We first observe that every infinite regular language has an infinite $\sim_e$-class.

▶ **Lemma 9.** *Let L be an infinite ω-regular language. Then there is an infinite $\sim_e$-class $X \in L/\sim_e$.*

Proof. Since L is an ω-regular language, by [2] it has the form $L = \bigcup_{1 \leq i \leq n} U_i V_i^\omega$ for some (finite-word) regular languages U_i, V_i which are not empty. We consider two cases.

Suppose that $V_i^\omega = \{v_i^\omega\}$ for each i, in which case L is countable. Since L is infinite there is an i such that $U_i v_i^\omega$ is also infinite.

Observe that the words in $U_i v_i^\omega$ need not be pairwise $\sim_e$-equivalent. For example consider $U = 1^*, v = 01$. Here it holds that $(01)^\omega \in Uv^\omega$ and $(10)^\omega \in Uv^\omega$ but $(01)^\omega \not\sim_e (10)^\omega$. Nevertheless, all $w \in U_i v_i^\omega$ fall into one of at most $|v_i|$ many $\sim_e$-classes and therefore $U_i v_i^\omega / \sim_e$ must contain an infinite class.

In the other case, there is a V_i which contains two words w, v such that $w^\omega \neq v^\omega$. Set $U_i' := U_i w^*$, then $U_i' v^\omega \subseteq (U_i V_i^*) V_i^\omega = U_i V_i^\omega$. The language $U_i' v^\omega$ is infinite. Otherwise the language $w^* v^\omega$ would also be finite and therefore $w^i v^\omega = w^j v^\omega$ for some $i \neq j$. But then $w^l = v^k$ for some $k, l \in \mathbb{N}$ and therefore $w^\omega = v^\omega$, contradiction. Since $U_i' v^\omega$ is infinite, we know from the first part of the proof that $U_i' v^\omega / \sim_e$ contains an infinite $\sim_e$-class. ◀

In the following we examine which elements of a structure can be encoded in the same end-class. For this undertaking, it is handy to consider elements of the original structure $\sim_e$-equivalent with respect to some given presentation.

▶ **Definition 10.** Let $\mathfrak{A}$ be a structure with ω-automatic presentation $(\mathcal{L}, \pi)$ and $\sim$ an equivalence relation on the domain of the presentation. We say $B \subseteq A$ is $(\sim, \mathcal{L}, \pi)$-equivalent or $\sim$-equivalent in $(\mathcal{L}, \pi)$ iff $B \subseteq \pi(X)$ for some $X \in L/\sim$.

If the presentation that is considered is clear from the context we will often just write that the set is $\sim$-equivalent without mentioning the presentation explicitly.

Observe that an equivalence relation $\sim$ on L does not need to induce an equivalence relation on A. Indeed, an element of A can have several encodings in $\mathcal{L}$ and can thus occur in the image of more than one $\sim$-class. However, if a set B is $\sim$-equivalent in $(\mathcal{L}, \pi)$, then there are encodings of the elements of B such that these codings are pairwise $\sim$-equivalent.

As a first application, we use Lemma 9 to make a quick observation about injective ω-automatic presentations.

▶ **Lemma 11.** *Let $\mathfrak{A}$ be an infinite structure. Then, for every injective ω-automatic presentation $(\mathcal{L}, \pi)$ of $\mathfrak{A}$, there is an infinite set $M \subseteq A$ that is $\sim_e$-equivalent in $(\mathcal{L}, \pi)$.*

Proof. Since A is infinite, L must also be infinite and therefore by Lemma 9 there must be an infinite class $X \in L/\sim_e$. Since π is injective, it follows that $\pi(X)$ is an infinite set that is $\sim_e$-equivalent in $(\mathcal{L}, \pi)$. ◀

The following lemma gives us a useful property of sets that are $\sim_e^m$-equivalent in an ω-automatic presentation. Intuitively, it states that the image of an $\sim_e^m$-equivalent set B under a definable k-ary function can always be partitioned into a constant number of sets which are also $\sim_e^m$-equivalent and this constant only depends on the presentation but not on B.

▶ **Lemma 12.** *Let $\mathfrak{A}$ be a structure with ω-automatic presentation $(\mathcal{L}, \pi)$ and let $f : A^{k+\ell} \to A$ be a function that is* FO*-definable in $\mathfrak{A}$. Then there is a constant q such that, for every $m \in \mathbb{N}$, every $(\sim_e^m, \mathcal{L}, \pi)$-equivalent subset $B \subseteq A$ and every tuple $\overline{a} \in A^\ell$, the image $f(B^k, \overline{a})$ admits a partition into q $(\sim_e^m, \mathcal{L}, \pi)$-equivalent sets $C_0, \ldots, C_{q-1}$.*

Proof. Let $\mathcal{A}$ be a Büchi automaton with states $Q = \{0,...,q-1\}$ that recognises f in $(\mathcal{L}, \pi)$.

First, choose a tuple of words $v_{\overline{a}} \in \pi^{-1}(\overline{a})$. Since B is $\sim_e^m$-equivalent in $(\mathcal{L}, \pi)$, there is a set $M \subseteq L$ of pairwise $\sim_e^m$-equivalent words such that $\pi \restriction M$ is a bijection between M and B. For every tuple $\overline{b} \in B^k$ we denote by $v_{\overline{b}}$ the unique tuple in M^k with $\pi(\overline{v}_{\overline{b}}) = \overline{b}$. Now choose, for every tuple $\overline{b} \in B^k$, a word $f_{\overline{b}} \in \pi^{-1}(f(\overline{b}, \overline{a}))$. This means that the word $\langle v_{\overline{b}}, v_{\overline{a}}, f_{\overline{b}} \rangle$ is accepted by $\mathcal{A}$ for every tuple $\overline{b} \in B^k$. Let $\rho_{\overline{b}}$ be an accepting run of $\mathcal{A}$ on $\langle v_{\overline{b}}, v_{\overline{a}}, f_{\overline{b}} \rangle$.

We obtain a partition of M^k into sets $M_0, \ldots, M_{q-1}$ by defining $M_i := \{v_{\overline{b}} : \rho_{\overline{b}}[m] = i\}$. For every nonempty M_i fix a tuple $v_{\overline{b}} \in M_i$. We will now show that, for any $\overline{d}$ with $v_{\overline{d}} \in M_i$, there is an encoding of $f(\overline{d}, \overline{a})$ that is $\sim_e^m$-equivalent to $f_{\overline{b}}$.

The intuition is that we can simply replace the tail of $\langle v_{\overline{d}}, v_{\overline{a}}, f_{\overline{d}} \rangle$ by the tail of $\langle v_{\overline{b}}, v_{\overline{a}}, f_{\overline{b}} \rangle$ and obtain a new word that is accepted by $\mathcal{A}$. This will give us a new encoding of $f(\overline{d}, \overline{a})$

that is $\sim_e^m$-equivalent to $f_{\overline{b}}$. More formally, for every such $\overline{d}$ it holds that $\rho_{\overline{d}}[0,m)\rho_{\overline{b}}[m,\omega)$ is an accepting run on

$$\begin{aligned}\langle v_{\overline{d}}, v_{\overline{a}}, f_{\overline{d}}\rangle[0,m)\langle v_{\overline{b}}, v_{\overline{a}}, f_{\overline{b}}\rangle[m,\omega) &= \langle v_{\overline{d}}, v_{\overline{a}}, f_{\overline{d}}\rangle[0,m)\langle v_{\overline{d}}, v_{\overline{a}}, f_{\overline{b}}\rangle[m,\omega)\\ &= \langle v_{\overline{d}}, v_{\overline{a}}, f_{\overline{d}}[0,m)f_{\overline{b}}[m,\omega)\rangle.\end{aligned}$$

This holds since $\langle v_{\overline{d}}\rangle \sim_e^m \langle v_{\overline{b}}\rangle$ and $\rho_{\overline{d}}[m] = \rho_{\overline{b}}[m] = i$. But since $\mathcal{A}$ recognises f in $\mathcal{L}, \pi$, it follows that $\pi(f_{\overline{d}}[0,m)f_{\overline{b}}[m,\omega)) = f(\overline{d},\overline{a})$.

So there is a $\sim_e^m$-class such that, for every $v_{\overline{d}} \in M_i$, the function value $f(\overline{d},\overline{a})$ has an encoding in this class. This implies that all the sets C_i' defined by $C_i' := \{f(\overline{b},\overline{a}) : v_{\overline{b}} \in M_i\}$ are $\sim_e^m$-equivalent in $(\mathcal{L},\pi)$. Obviously it holds that $\bigcup_{0\le i\le q-1} C_i' = f(B^k,\overline{a})$. We can now simply define C_i to be the set $C_i' - \bigcup_{0\le j<i} C_j'$ and obtain a partition $C_0,\ldots,C_{q-1}$ of $f(B^k,\overline{a})$ with the desired properties. ◀

From this result we infer that, in injectively ω-automatic structures, no FO-definable k-ary function can guarantee that the image of a finite set is always much larger than the set itself. We can make this precise by the notion of the minimal image size.

▶ **Definition 13.** The *minimal image size* of a function $f : A^k \to A$ over an infinite set A, $MIS_f : \mathbb{N} \to \mathbb{N}$, is given by $MIS_f(n) = \min\{|f(X^k)| : X \subseteq A, |X| = n\}$.

We now show that, for injectively presentable structures, the minimal image size of every FO-definable function grows at most linearly with n.

▶ **Lemma 14.** *Let $\mathfrak{A}$ be an infinite structure with an injective ω-automatic presentation. Then, for every* FO*-definable function f, it holds that $MIS_f(n) = \mathcal{O}(n)$.*

Proof. Suppose there is an injective automatic presentation $(\mathcal{L},\pi)$ of an infinite structure with FOdefinable function $f : A^k \to A$ such that MIS_f grows in a superlinear way.

Let $\mathcal{A} = (Q, \Sigma^{k+1}, q_0, \Delta, F)$ be a Büchi automaton that recognizes f in $(\mathcal{L},\pi)$ and c the constant from Lemma 12 with respect to f and $(\mathcal{L},\pi)$. Now choose n such that $MIS_f(n) > c|\Sigma| \cdot n$. This is possible since MIS_f grows in a superlinear way. By Lemma 11 there is an infinite set $M \subseteq A$ that is $\sim_e$-equivalent in $(\mathcal{L},\pi)$. Therefore we can choose m such that there is a $(\sim_e^m, \mathcal{L}, \pi)$-equivalent set of size at least n. Choose m minimal and let N be a $\sim_e^m$-equivalent set of maximal size. Then $n \le |N| \le |\Sigma|n$, otherwise we could have chosen m smaller.

Observe that if $MIS_f(n) = a$ then for all sets X with $|X| \ge n$ it holds that $|f(X)| \ge a$. By Lemma 12, $f(N^k)$ can be partitioned into c $\sim_e^m$-equivalent sets. One of these sets has size at least $|f(N^k)|/c > |\Sigma|n \ge |N|$. But this contradicts the maximality of N among all $\sim_e^m$-equivalent sets. ◀

Since pairing functions grow in a quadratic way, we obtain the following corollary.

▶ **Corollary 15.** *No infinite structure with an* FO*-definable pairing function admits an injective ω-automatic presentation.*

5 Fields of Characteristic 0

Every field of characteristic 0 contains, in a canonic way, a copy of $(\mathbb{N}, +, \cdot)$. We first show that, whenever a field of characteristic 0 admits an ω-automatic presentation, the size of every $\sim_e$-equivalent set of natural numbers is bounded by a constant.

▶ **Lemma 16.** *Let $(K,+,\cdot)$ be a field of characteristic 0 with an ω-automatic presentation $(\mathcal{K},\pi)$. Then there exists a constant q such that no $(\sim_e,\mathcal{K},\pi)$-equivalent set N of natural numbers has size $|N|>q$.*

Proof. Consider the function $p : K \times K \to K$ defined by

$$p(x,y) := \frac{(x+y)(x+y+1)}{2} + y.$$

Note that p restricted to the natural numbers is the Cantor pairing function. Now let q be the constant from Lemma 12 associated with the function p with respect to $(\mathcal{K},\pi)$. Towards a contradiction, assume there exist sets of natural numbers that are $\sim_e$-equivalent in $(\mathcal{K},\pi)$ and have more than q elements. This implies that, for some m, there exists a $(\sim_e^m,\mathcal{K},\pi)$-equivalent set of natural numbers of size larger than q.

Fix such an m and let N be one of the largest such sets, i.e. such that $|N| \geq |M|$ for every $M \subseteq \mathbb{N}$ that is $\sim_e^m$-equivalent in $(\mathcal{K},\pi)$. Now apply Lemma 12 and obtain a partition $M_1,\ldots,M_q$ of $p(N^2) \subseteq \mathbb{N}$ such that every M_i is $\sim_e^m$-equivalent in $(\mathcal{K},\pi)$. Since p is a pairing function on $\mathbb{N}$, we have

$$|p(N^2)| = |N|^2 > q \cdot |N|.$$

It follows, by the pigeonhole principle, that there must be a M_i of size at least $|N|^2/q > |N|$. But, since M_i is a set of natural numbers that is $(\sim_e^m,\mathcal{K},\pi)$-equivalent, this contradicts the choice of N as one of the largest sets. ◀

We will now generalise this statement from sets of natural numbers to arbitrary subsets of the field. The key here is the use Freiman's Theorem and Lemma 8 derived from it.

▶ **Lemma 3.** Whenever a field $(K,+,\cdot)$ of characteristic 0 admits an ω-automatic presentation $(\mathcal{K},\pi)$ then the size of all $\sim_e$-equivalent subsets of $\mathcal{K}$ is bounded by a constant r.

Proof. We show that every set $M \subseteq K$ that is $\sim_e$-equivalent in $\mathcal{K},\pi$ has size $|M| \leq r$.

Let q_1 be the constant q from Lemma 12 with respect to the function $(x-y_1)/y_2$, let q_2 be the constant q from Lemma 12 with respect to $+$, and let q_3 be the constant q from Lemma 16. We then set $c := \max\{q_1,q_2,q_3\}$ and let k,d be the constants from Freiman's Theorem with respect to c. Finally, we choose $r = c^{2d} \cdot k^d$.

Suppose there is a set M that is $\sim_e$-equivalent in $(\mathcal{K},\pi)$ with $|M| > r$. Then we can also choose an m such that there are $\sim_e^m$-equivalent sets of size larger than r. Choose such a $\sim_e^m$-equivalent set M of maximal size.

It is easy to see that $|M+M| \leq c|M|$. Otherwise we could argue analogously to the proof of Lemma 16 and obtain a set C that is larger than M and also $\sim_e^m$-equivalent in $(\mathcal{K},\pi)$.

By Lemma 8 there is an arithmetic progression $P = \{p_0 + mp_1 : m < n, m \in \mathbb{N}\}$ such that $N := M \cap P$ has size at least $\frac{\sqrt[d]{|M|}}{k}$. Now, by Lemma 12, we can partition the set $\{(a-p_0)/p_1 : a \in N\} \subseteq \mathbb{N}$ into at most c sets and obtain a $\sim_e^m$-equivalent set of natural numbers of size at least

$$\frac{\sqrt[d]{|M|}/k}{c} > \frac{\sqrt[d]{c^{2d}k^d}}{ck} = c.$$

But this contradicts Lemma 16. ◀

For injective presentations, Lemma 11 ensures infinite $\sim_e$-equivalent sets, but Lemma 3 forbids such sets, and therefore we directly obtain Theorem 1. For Theorem 2, more work is necessary. In the next section we show a result that will enable us to lift the above argument from injective presentations to general ones for fields with a definable linear order.

6 ω-Automatic Linear Orders

For uncountable fields of characteristic 0 with a definable linear order, we transfer the result of the previous section from injective presentations to general ω-automatic ones. The problem that we face if we consider non-injective presentations $(\mathcal{L}, \pi)$ is that we cannot simply infer that there are infinite $\sim_e$-equivalent sets. For example, $\sim_e$ is an ω-automatic equivalence relation and identifies all ultimately equal words.

But, as we will see, this cannot happen for ω-automatic presentations of uncountable linear orders. To prove this, it suffices to show that every ω-automatic presentation of an uncountable linear order $(A, <)$ contains an infinite regular subset such that the presentation, restricted to this subset, is an injective presentation of a suborder of $(A, <)$. In [9] Kuske had already shown that $(\{0,1\}^\omega, <_{lex})$ is embeddable into any ω-automatic uncountable linear order. He constructs, from a given ω-automatic presentation of such an order, a subpresentation that is a presentation of $(\{0,1\}^\omega, <_{lex})$. This subpresentation, however, is not ω-automatic: its domain is the complement of a language $\bigcup_{i\leq n} V_i U_i^\omega$ where the V_i are context free and and the U_i are regular. In particular, his presentations do not contain two $\sim_e$-equivalent words.

We will therefore present here a strengthening of Kuske's result. We show that every automatic presentation of an uncountable linear order contains an injective automatic presentation of $(\{0,1\}^\omega, <_{lex})$.

The main techniques originate from [6]. For a given ω-automatic presentation $(L, \approx, <)$ we construct finite words u, v_0, v_1 such that $u\{v_0, v_1\}^\omega \subseteq L$ and, for any two words $\alpha, \beta \in \{0,1\}^\omega$, it holds that

$$uv_{\alpha[0]}v_{\alpha[1]}v_{\alpha[2]}\ldots < uv_{\beta[0]}v_{\beta[1]}v_{\beta[2]}\ldots \text{ if, and only if, } \alpha <_{lex} \beta.$$

This means v_0 encodes 0, v_1 encodes 1 and the language $u\{v_0, v_1\}^\omega$ encodes $\{0,1\}^\omega$ in a natural way.

To handle these constructions, it is convenient to use an algebraic characterisation of ω-regular languages. More precisely, we use that a language L is ω-automatic if, and only if, there is an ω-semigroup morphism $g : \Sigma^\infty \to S$ from the free ω-semigroup $\Sigma^\infty = (\Sigma^*, \Sigma^\omega)$ into some finite ω-semigroup $S = (S_f, S_\omega)$ with $g^{-1}(g(L)) = L$. A broader introduction to the topic can be found in [10]. The advantage of having such a morphism g is that we know, whenever $g(u) = g(v)$ for two words u and v, that $u \in L$ if, and only if, $v \in L$. With this property it is much easier to ensure that the elements we construct have the properties that we want than by cutting apart and rearranging runs of automata.

▶ **Theorem 17.** *For every ω-automatic presentation $(L, \approx, <)$ of an uncountable linear order there exists a subset L' of L such that $(L', <\cap(L')^2)$ is an injective ω-automatic presentation of $(\{0,1\}^\omega, <_{lex})$.*

Proof. Let $\mathcal{L} = (L, \approx, <)$ be an ω-automatic presentation of an uncountable linear order. Since $\mathcal{L}$ is automatic, for each $\delta \in \{L, \approx, <\}$ there are ω-semigroup morphisms to finite ω-semigroups $S_\delta = (S_f^\delta, S_\omega^\delta)$:

$$()^L : \Sigma^\infty \to S_L, \quad ()^\approx : (\Sigma \times \Sigma)^\infty \to S_\approx, \quad ()^< : (\Sigma \times \Sigma)^\infty \to S_<$$

that recognise the corresponding relations. We set $F_L := (L)^L$ and for $\sim \in \{\approx, <\}$ we define $F_\sim := (\{\langle u, v\rangle : u \sim v\})^\sim$. We define c to be the size of the largest ω-semigroup among $S_L, S_\approx, S_<$, set $C := c^3$ and k as the least common multiple of the exponents of these semigroups.

As mentioned before, our goal is to show that there are $u, v_0, v_1 \in \Sigma^+$ with $v_0 \neq v_1$, $|v_0| = |v_1|$ such that $u\{v_0, v_1\}^\omega \subseteq L$ and for all $\alpha, \beta \in \{0,1\}^\omega$ it holds that $uv_\alpha < uv_\beta$ iff $\alpha <_{lex} \beta$ (here v_α stands for $v_{\alpha[0]}v_{\alpha[1]}v_{\alpha[2]}\ldots$). This obviously means, that $\mathcal{L}$ restricted to $u\{v_0, v_1\}^\omega$ is an injective ω-automatic presentation of $(\{0,1\}^\omega, <_{lex})$.

Before we start we need to fix some notions. We call an infinite set $H = \{h_0 < h_1 < h_2 < \ldots\} \subseteq \mathbb{N}$ a factorisation. For a given ω-semigroup morphism $g : \Sigma^\infty \to S$ and $w \in \Sigma^\omega$, we say that H is a g, e-homogeneous factorisation of w if $g(w[h_i, h_{i+1})) = e$ for all $i \in \mathbb{N}$.

The following lemma is basically a reformulation of Ramsey's theorem for countably infinite graphs to the language of ω-semigroup morphisms. It ensures the existence of factorisations that are homogeneous for several words and morphisms.

▶ **Lemma 18.** *For $i \in \{1, \ldots, n\}$ let $h_i : \Sigma_i^\infty \to S_i$ be a morphism into a finite ω-semigroup S_i and w_i a word over the corresponding alphabet. There exists a $G = \{g_1 < g_2 < g_3 \ldots\} \subseteq \mathbb{N}$ such that, for every i, G is an h_i,e_i-homogeneous factorisation of w_i for some idempotent semigroup element $e_i \in S_i$.*

Proof. This lemma is more or less a direct consequence of Ramsey's theorem: We colour every $\{k, l\} \subset \mathbb{N}$, $k < l$ with the tuple of semigroup elements $[h_i(w_i[k, l))]_{1 \leq i \leq n}$. From Ramsey's theorem we obtain a $G = \{g_1 < g_2 < g_3 \ldots\}$ such that all $\{g_i, g_j\}$, $i \neq j$ have the same colour. Having set $e_i := h_i(w_i[g_1, g_2))$, G obviously is a h_i,e_i-homogeneous factorisation of w_i and e_i is idempotent since $e_ie_i = h_i(w_i[g_1, g_2))h_i(w_i[g_2, g_3)) = h_i(w_i[g_1, g_3)) = e_i$. ◀

Now we are ready to start the construction. As a first step, we show that there are two words x_0 and x_1 and a factorisation H such that, regarding H, x_0 and x_1 are mapped to the same elements under every mentioned morphism. Later on we will use the obtained words to cut out suitable candidates for u, v_0 and v_1.

▶ **Lemma 19.** *There exist $x_0, x_1 \in L$ with $[x_i]_{\sim_e} \cap [x_{1-i}]_\approx = \emptyset$ and a factorisation $H = \{h_1 < h_2 < h_3 < \ldots\}$ such that, for some idempotent $e_L \in S_L$, H is a $()^L, e_L$-homogeneous factorisation of x_0 and x_1. For $\delta \in \{\approx, <\}$ there are idempotent e_δ, e_δ^{01} and e_δ^{10} such that*

- *H is a $()^\delta, e_\delta$-homogeneous factorisation of $\langle x_0, x_0\rangle$ and $\langle x_1, x_1\rangle$ and*
- *H is a $()^\delta, e_\delta^{01}$-homogeneous factorisation of $\langle x_0, x_1\rangle$ and*
- *H is a $()^\delta, e_\delta^{10}$-homogeneous factorisation of $\langle x_1, x_0\rangle$.*

Proof Sketch. Since $\mathcal{L}$ is a presentation of an uncountable structure, there are elements $y_0, \ldots, y_C$ such that $[y_i]_{\sim_e} \cap [y_j]_\approx = \emptyset$ for $i \neq j$. With Lemma 18 we obtain an H that is a homogeneous factorisation of all y_i and respectively all $\langle y_i, y_j\rangle$ under all mentioned morphisms. Since we have more elements y_i than triples $(a, b, c) \in S_L \times S_\approx \times S_<$ there must be some $i \neq j$ such that H is a $()^L, e_L$-homogeneous factorisation of y_i and y_j and for $\delta \in \{\approx, <\}$ a $()^\delta, e_\delta$-homogeneous factorisation of $\langle y_i, y_i\rangle$ and $\langle y_j, y_j\rangle$. ◀

We may also assume that H is coarse enough that $x_0[h_l, h_{l+1}) \neq x_1[h_l, h_{l+1})$ for all $l \in \mathbb{N}$. We need to modify x_0, x_1 a bit to ensure all the properties required later.

▶ **Lemma 20.** *There exist $y_0, y_1 \in L_A$ with $y_0 \not\approx y_1$ and a factorisation $G = \{g_1 < g_2 < g_3 < \ldots\}$ with the following properties:*

- *$y_0[0, g_1) = y_1[0, g_1)$ and $y_0[g_i, g_{i+1}) \neq y_1[g_i, g_{i+1})$ for all $i \in \mathbb{N}$.*
- *for $\delta \in \{\approx, <\}$ there is an element $\rightarrow_\delta \in S_f^\delta$ and idempotents $\square_\delta, \uparrow_\delta, \downarrow_\delta \in S_f^\delta$ such that*
  - *$\langle y_0, y_0\rangle[0, g_1)^\delta = \rightarrow_\delta$,*
  - *$\langle y_0, y_0\rangle[g_i, g_{i+1})^\delta = \langle y_1, y_1\rangle[g_i, g_{i+1})^\delta = \square_\delta$,*

- $\langle y_0, y_1\rangle[g_i, g_{i+1})^\delta = \uparrow_\delta$,
- $\langle y_1, y_0\rangle[g_i, g_{i+1})^\delta = \downarrow_\delta$ *and*
- $\rightarrow_\delta, \uparrow_\delta$ *and* $\downarrow_\delta$ *absorb* $\Box_\delta$.

Proof Sktech. We first construct y_0 and y_1 and then show that these have the desired properties. We define y_0 as $y_0 := x_1[0, h_2)x_0[h_2, \omega)$ and y_1 by

$$\begin{aligned} y_1[0, h_2) &:= x_1[0, h_2) \text{ and} \\ y_1[h_{2l}, h_{2l+2}) &:= x_1[h_{2l}, h_{2l+1})x_0[h_{2l+1}, h_{2l+2}) \text{ for } l \geq 1. \end{aligned}$$

We set $G = \{h_{2kl+2} : l \in \mathbb{N}\}$ (remember k is the least common multiple of the exponents of the involved semigroups). It holds that $\langle y_0, y_1\rangle^{\approx} = \langle y_1, x_1\rangle^{\approx}$. So if $y_0 \approx y_1$ then also $y_1 \approx x_1$ and therefore $y_0 \approx x_1$. But $y_0 \sim_e x_0$ in contradiction to $[x_0]_{\sim_e} \cap [x_1]_{\approx} = \emptyset$. That the other postulated properties hold can be established by straightforward calculations. ◀

Now that we have y_0 and y_1, we are ready to construct u, v_0 and v_1. We set

$$u := y_1[0, g_0), \qquad v_0 := y_0[g_0, g_1) \text{ and} \qquad v_1 := y_1[g_0, g_1).$$

From this definition we immediately get for $\delta \in \{\approx, <\}$:

$$\langle u, u\rangle^\delta = \rightarrow_\delta , \quad \langle v_0, v_0\rangle^\delta = \langle v_1, v_1\rangle^\delta = \Box_\delta, \quad \langle v_0, v_1\rangle^\delta = \uparrow_\delta \text{ and,} \quad \langle v_1, v_0\rangle^\delta = \downarrow_\delta .$$

In the following we will omit the subscripts and just write $\rightarrow, \uparrow, \downarrow$ and $\Box$ since it will be obvious from the context which δ is meant.

We will now show that $u\{v_0, v_1\}^\omega$ has all the properties that were announced at the beginning of the proof.

▶ **Lemma 21.** $u\{v_0, v_1\}^\omega \subseteq L$

Proof. Let α be any sequence from $\{0, 1\}^\omega$.

$$(uv_\alpha)^L = y_1[0, g_0)^L (y_{\alpha[i]}[g_0, g_1)^L)_{i\in\mathbb{N}} = x_1[0, g_0)^L (e_L)^\omega = (x_1)^L \in F_L$$

This means every uv_α is in L and therefore $u\{v_0, v_1\}^\omega \subseteq L$. ◀

Next we show that at least some words from $u\{v_0, v_1\}^\omega$ do encode distinct elements.

▶ **Lemma 22.** $\rightarrow (\uparrow\downarrow)^\omega \notin F_{\approx}$.

Proof. First we see that $\rightarrow\uparrow^\omega \notin F_{\approx}$ since $\langle y_0, y_1\rangle^{\approx} = \rightarrow\uparrow^\omega$ and $y_0 \not\approx y_1$. We will make use of the transitivity of $\approx$ to show that also $\rightarrow (\uparrow\downarrow)^\omega \notin F_{\approx}$. Suppose $\rightarrow (\uparrow\downarrow)^\omega \in F_{\approx}$, then consider the words $u(v_0v_1v_0)^\omega$,$u(v_1v_0v_1)^\omega$ and $u(v_1v_1v_0)^\omega$. We have

$\langle u(v_0v_1v_0)^\omega, u(v_1v_0v_1)^\omega\rangle^{\approx} = \rightarrow (\uparrow\downarrow\uparrow)^\omega = \rightarrow (\uparrow\downarrow)^\omega$ and $\langle u(v_1v_0v_1)^\omega, u(v_1v_1v_0)^\omega\rangle^{\approx} = \rightarrow (\Box\uparrow\downarrow)^\omega = \rightarrow (\uparrow\downarrow)^\omega$.

Hence $u(v_0v_1v_0)^\omega \approx u(v_1v_0v_1)^\omega$ and $u(v_1v_0v_1)^\omega \approx u(v_1v_1v_0)^\omega$ and so by transitivity $u(v_0v_1v_0)^\omega \approx u(v_1v_1v_0)^\omega$, but $\langle u(v_0v_1v_0)^\omega, u(v_1v_1v_0)^\omega\rangle^{\approx} = \rightarrow (\uparrow \Box\Box)^\omega = \rightarrow\uparrow^\omega \notin F_{\approx}$, contradiction. ◀

We conclude our proof by showing that $<$ is indeed the lexicographic order on $u\{v_0, v_1\}^\omega$.

▶ **Lemma 23.** *Either it holds for every* $\alpha \neq \beta \in \{0, 1\}^\omega$ $uv_\alpha < uv_\beta$ *iff* $\alpha <_{lex} \beta$ *or it holds for every* $\alpha \neq \beta \in \{0, 1\}^\omega$ $uv_\alpha < uv_\beta$ *iff* $\beta <_{lex} \alpha$.

Proof Sketch. First observe that $u(v_0v_1)^\omega \not\approx u(v_1v_0)^\omega$ since $\langle u(v_0v_1)^\omega, u(v_1v_0)^\omega\rangle^{\approx} = \rightarrow (\uparrow\downarrow)^\omega \notin F_{\approx}$. Therefore either $u(v_0v_1)^\omega < u(v_1v_0)^\omega$ or $u(v_1v_0)^\omega < u(v_0v_1)^\omega$ holds. In the first case, we can show, for every $\alpha \neq \beta \in \{0,1\}^\omega$, that $uv_\alpha < uv_\beta$ iff $\alpha <_{lex} \beta$. The other case leads to the other part of the lemma's statement and can be shown analogously.

Using the idempotence and absorption properties of $\uparrow, \downarrow$ and $\rightarrow$ it is possible to write all elements $\langle uv_\alpha, uv_\beta\rangle^<$ as infinite products with no consecutive occurrences of two $\uparrow, \downarrow$ or $\square$ (except for infinite occurrences at the end). For every such infinite product ρ that originates from the image of a word $\langle uv_\alpha, uv_\beta\rangle$ with $\alpha <_{lex} \beta$, it is possible to find words $w_1, w_2, w_3 \in u\{v_0, v_1\}^\omega$ with $\langle w_1, w_2\rangle^< = \langle w_2, w_3\rangle^< = \rightarrow (\uparrow\downarrow)^\omega$ and $\langle w_1, w_3\rangle^< = \rho$. But, by transitivity of $<$, this implies $\rho \in F_<$ and therefore we get that $u\{v_0, v_1\}^\omega$ is ordered as desired. ◀

Taking all together, we get that $\mathcal{L}$ restricted to $u\{v_0, v_1\}^\omega$ is an injective ω-automatic presentation of $(\{0,1\}^\omega, <_{lex})$, which completes the proof of Theorem 17. ◀

Combining the above Theorem 17 and Lemma 3 proves Theorem 2 for uncountable fields of characteristic 0 with definable linear orders. Since countable ω-automatic structures have injective presentations [6], these are covered by Theorem 1, and Theorem 2 follows.

References

1 A. Blumensath and E. Grädel. Automatic structures. In *Proc. of LICS '00*, pages 51–62, 2000.

2 J. R. Büchi. On decision method in restricted second order arithmetic. In *Proc. of the International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11, 1962.

3 G. Freiman. Foundations of a structural theory of set addition (in russian). *Gos. Ped. Inst. Kazan*, 1966.

4 B. Green. Structure theory of set addition. Unpublished lecture notes, available from `http://www.dpmms.cam.ac.uk/~bjg23/notes.html`, 2002.

5 G. Hjorth, B. Khoussainov, A. Montalbán, and A. Nies. From automatic structures to Borel structures. In *Proc. of LICS'08*, pages 431–441. IEEE Computer Society, 2008.

6 Ł. Kaiser, S. Rubin, and V. Bárány. Cardinality and counting quantifiers on ω-automatic structures. In *Proc. of STACS'08*, volume 1 of *LIPIcs*, pages 385–396. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Germany, 2008.

7 Khoussainov and Nerode. Automatic presentations of structures. In *Proc. of LCC: International Workshop on Logic and Computational Complexity*, 1994.

8 B. Khoussainov, A. Nies, S. Rubin, and F. Stephan. Automatic structures: Richness and limitations. *Logical Methods in Computer Science*, 3(2), 2007.

9 D. Kuske. Is Ramsey's theorem omega-automatic? In *Proc. of STACS'10*, volume 5 of *LIPIcs*, pages 537–548. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2010.

10 D. Perrin and J.-E. Pin. Semigroups and automata on infinite words. In *Semigroups, Formal Languages and Groups*, NATO Advanced Study Institute, pages 49–72. Kluwer academic publishers, 1995.

11 M. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the AMS*, 141:1–35, 1969.

12 I. Ruzsa. Generalized arithmetical progressions and sumsets. *Acta Math. Hungar.*, 65:379–388, 1994.

13 A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, 1951.

14 T. Tsankov. The additive group of the rationals does not have an automatic presentation. *Journal for Symbolic Logic (to appear)*, 2011.

The Limits of Decidability for First Order Logic on CPDA Graphs*

Christopher H. Broadbent

LIAFA, Université Paris Diderot-Paris 7 and CNRS
Case 7014 F-75205 Paris Cedex 13, France
broadbent@liafa.jussieu.fr

Abstract

Higher-order pushdown automata (n-PDA) are abstract machines equipped with a nested 'stack of stacks of stacks'. Collapsible pushdown automata (n-CPDA) extend these devices by adding 'links' to the stack and are equi-expressive for tree generation with simply typed λY terms. Whilst the configuration graphs of HOPDA are well understood, relatively little is known about the CPDA graphs. The order-2 CPDA graphs already have undecidable MSO theories but it was only recently shown by Kartzow [9] that first-order logic is decidable at the second level. In this paper we show the surprising result that first-order logic ceases to be decidable at order-3 and above. We delimit the fragments of the decision problem to which our undecidability result applies in terms of quantifer alternation and the orders of CPDA links used. Additionally we exhibit a natural sub-hierarchy enjoying limited decidability.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.1 Mathematical Logic

Keywords and phrases Collapsible Pushdown Automata, First Order Logic, Logical Reflection

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.589

1 Introduction

Higher-order pushdown automata generalise traditional pushdown automata by allowing the stack to contain other stacks rather than just atomic elements. These devices are closely related to *recursion schemes*, which are essentially simply typed λY terms that generate a single infinite tree. Enjoying decidable μ-calculus theories, the class of trees generated by recursion schemes shows a lot of promise as a model for verifying higher-order functional programs [12, 13]. Unfortunately n-PDA expressively coincide with order-n recursion schemes only when the latter satisfy a property called *safety* [10], unsafe recursion schemes are strictly more expressive [15, 14]. Hence a more powerful automaton is needed, which motivates order-n *collapsible pushdown automata* (n-CPDA) [7]. Inspired by *panic automata* [11], which can be viewed as the special case at order-2, atomic elements in collapsible stacks eminate 'links' that target a component of the stack further below.

We concern ourselves here with configuration *graphs* of these automata with *reachable* states as vertices and transitions as edges. It is particularly fruitful to additionally consider the 'ϵ-closures' of such graphs, whose edges consist of an unbounded number of transitions rather than just single steps. The ϵ-closures of n-PDA graphs form precisely the *Caucal Hierarchy* [6, 3, 5], which is defined independently in terms of graph transformations. This deep result has as a consequence that every n-PDA graph has decidable MSO theory.

* The author is funded by La Fondation Sciences Mathématiques de Paris.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 589–600
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

	2-CPDA /w ϵ-cl.	3_2-CPDA	3_2-CPDA /w ϵ-cl.	n_n-CPDA (/w ϵ-cl.) $n \geq 3$	$n_{n,(n-1)}$-CPDA $n \geq 4$ /wo ϵ-cl. $n \geq 3$ /w ϵ-cl.	n_m-CPDA $n \geq 4$, $m \leq n-2$	n-PDA /w ϵ-cl. all n
Σ_1	Dec [9]	Dec [1]	Dec [1]	**Dec**	?	**Und**	Dec [5]
Σ_2	Dec [9]	Dec [1]	**Und**	**Und**	**Und**	**Und**	Dec [5]
FO	Dec [9]	Dec [1]	**Und**	**Und**	**Und**	**Und**	Dec [5]
MSO	Und [7]	Und [7]	Und [7]	Und [7]	Und [7]	Und [7]	Dec [5]
μ-c.	Dec [11]	Dec [7]	Dec [7]	Dec [7]	Dec [7]	Dec [7]	Dec [10]

Table 1 Summary of (un)decidability results known to date. Those in bold are from this paper. The notation '$n_{m,m'}$-CPDA' means an n-CPDA that only uses links of orders m and m'.

Unfortunately there is even a 2-CPDA graph that has *un*decidable MSO theory [7]. Nevertheless the local nature of first-order logic meant it was widely assumed the first-order theories would still enjoy decidability and indeed Kartzow saw that the ϵ-closures of 2-CPDA graphs are *tree automatic* [9] and so do indeed have decidable first-order theory. Our first contribution is to show that Kartzow's result cannot be extended to higher-orders in full generality. At order-3 we get undecidability when we consider Σ_2-sentences, namely those with quantifier alternation of the form $\forall\exists$. If we allow order-3 links we do not even need ϵ-closure for this. This result is interesting in itself, but we also gain some insight into what links 'mean' in terms of 3-CPDA *graphs*. On the one hand links can act as 'place holders' that allow first-order logic to compare internal components of a single order-3 stack rather than just two order-3 stacks in their entirety. This is the core of the undecidability result, which goes via a reduction from Post's Correspondence Problem [16]. Additionally order-3 links provide edges in the graph that are 'non-local' in nature, allowing ϵ-closure to be eliminated as a requirement for undecidability. At order-4 we get that even the Σ_1-theory is undecidable—*viz.* the theory consisting of sentences *without any* quantifier alternation.

Our second contribution introduces a technique to tackle the Σ_1-theories of CPDA graphs. Making use of *logical reflection* [2], which enables CPDA to 'know' which μ-calculus sentences they satisfy at any given point, we define a notion of *monotonic CPDA* that constructs all its reachable configurations without ever destroying an order-$(n-1)$ stack. This has some parallels with Carayol's work on canonical sequences of operations witnessing the constructibility of n-PDA stacks [4]. If an n-CPDA only has 'order-n links', monotonicity allows us to eliminate them, thereby producing an n-PDA with decidable MSO theory, and leading to the decidability of the Σ_1 theory of the original n-CPDA. This is a graph analogue of Aehlig *et al.'s* [8] in which links are eliminated from 2-CPDA to produce an equivalent word-generating 2-PDA. This decidability result is the first about a fragment of first-order logic on n-CPDA at *all* orders and shows restricting links can recover some decidability.

We emphasise that even the undecidability results without ϵ-closure still require a restriction of the domain to configurations *reachable from the origin*. Unlike word-automatic structures, however, this undecidability is non-trivial, as indicated by the positive results.

2 Preliminaries

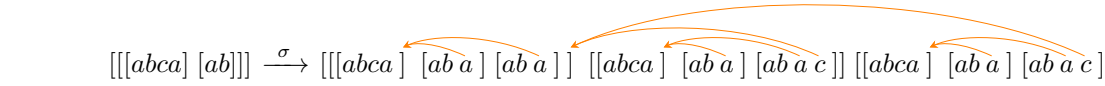
2.1 Higher-Order Stacks

Let us fix a stack-alphabet Γ. For higher-order automata this alphabet must be finite, but it is convenient for definitions to allow it to be infinite. An *order*-1 stack over Γ is just a string of

the form $[\ \gamma\ ]$ where $\gamma \in \Gamma^*$. Let us refer to the set of order-1 stacks over Γ as $stack_1(\Gamma)$. For $n \in \mathbb{N}$ the set of *order*-$(n+1)$ stacks is: $stack_{n+1}(\Gamma) := stack_1(stack_n(\Gamma))$. We allow the following operations on an order-1 stack s for every $a \in \Gamma$: $push_1^a([\ a_1 \cdots a_m\ ]) := [\ a_1 \cdots a_m a\ ]$, $pop_1([\ a_1 \cdots a_m a_{m+1}\ ]) := [\ a_1 \cdots a_m\ ]$, $nop(s) := s$. We allow the following *order*-$(n+1)$ *operations* on an order-$(n+1)$ stack s, where θ is any *order-n operation*: $push_{n+1}([\ s_1 \cdots s_m\ ]) := [\ s_1 \cdots s_m s_m\ ]$, $pop_{n+1}([\ s_1 \cdots s_m s_{m+1}\ ]) := [\ s_1 \cdots s_m\ ]$, $\theta([\ s_1 \cdots s_m\ ]) := [\ s_1 \cdots \theta(s_m)\ ]$. Where s is an $(n+1)$-stack we write $top_{n+1}(s)$ to denote the top-most (right-most) order-n stack (abusing notation with $top_{n+1}(s) := s$ if s is an n-stack) and $top_1(s)$ as the top atomic element. Let us also write $s \sqsubseteq t$ when $s = [\ s_1 \cdots s_m\ ]$ and t is of the form $[\ s_1 \cdots\ s_{m'}\ ]$ for $m' \leq m$, $s \sqsubset t$ when additionally $s \neq t$ and define $|s| := m$.

2.2 Collapsible Pushdown Stacks

A $push_1^{a,k}$ operation in a CPDA attaches a *k-link* from the newly created a element that points to the k-stack below. The targets of these links are preserved during higher-order *push* operations. Consider the sequence $\sigma = push_1^{a,2}; push_2; push_3; push_1^{c,3}; push_3$, then:



We offer fine control over the orders of links that a collapsible stack may contain; an order-n_S stack is an order-n stack equipped with order-i links for each $i \in S$. Formally the *S-collapsible pushdown alphabet* $\Gamma^{[S]}$ (for $S \subseteq \mathbb{N}$) induced by an alphabet Γ is the set $\Gamma \times S \times \mathbb{N}$. The set of order-$n_S$ collapsible stacks $stack_{n_S}^{\mathcal{C}}(\Gamma)$ is defined by: $stack_{n_S}^{\mathcal{C}}(\Gamma) := stack_n(\Gamma^{[S]})$. An atomic element $(a, l, p) \in \Gamma^{[S]}$ has label a with a link of order l. If $l < n$ the target of a link is the pth $(l-1)$-stack of the l-stack in which the element resides.

We thus introduce the operations $push_1^{a,l}(s) := push_1^{(a,l,|top_{l+1}(s)|-1)}(s)$ and $collapse(s) := pop_l^{|top_{l+1}(s)|-p}$ where $top_1(s) = (a, l, p)$ and θ^m represents the operation θ iterated m times. From now on we will abuse notation and consider $top_1(s) := a$. Write Θ_{n_S} to denote the set of order-n_S collapsible stack operations. Write $\bot_n$ for the empty n-stack.

2.3 The Automata and their Graphs

Let $n \in \mathbb{N}$ and let $S \subseteq [1..n]$. An n_S-CPDA (order-n_S collapsible pushdown automaton) $\mathcal{A}$ is a tuple: $\langle\ \Sigma, \Pi, Q, q_0, \Gamma, R_{a_1}, R_{a_2}, \ldots, R_{a_r}, P_{b_1}, P_{b_2}, \ldots, P_{b_{r'}}\ \rangle$ where Σ is a finite set of transition labels $\{a_1, a_2, \ldots, a_r\}$; Π is a finite set of configuration labels $\{b_1, b_2, \ldots, b_{r'}\}$; Q is a finite set of control-states; $q_0 \in Q$ is an initial control-state; Γ is a finite stack alphabet; each R_{a_i} is the a_i-labelled transition relation with $R_{a_i} \subseteq Q \times \Gamma \times \Theta_{n_S} \times Q$; each P_{b_i} is the b_i-labelled unary predicate specified by $P_{b_i} \subseteq Q \times \Gamma$. Remaining consistent with the definitions in the literature, an n-CPDA is an $n_{[n..2]}$-CPDA and an n-PDA is an $n_\emptyset$-CPDA.

A *configuration* of an n_S-CPDA $\mathcal{A}$ is a pair (q, s) where q is a control-state and s is an n_S-stack. Such a configuration *satisfies* the predicate $b_i \in \Pi$ just in case $(q, top_1(s)) \in P_{b_i}$. We say $\mathcal{A}$ can *a_i-transition from (q, s) to $(q', \theta(s))$*, written $(q, s) \xrightarrow{a_i} (q', \theta(a))$, iff $(q, top_1(s), \theta, q') \in R_{a_i}$. Let us further say that *(q', s') can be reached from (q, s) in $\mathcal{A}$ with path labeled in $\mathcal{L}$* for some $\mathcal{L} \subseteq \Sigma^*$ iff $(q, s) \xrightarrow{a_{i_1}} (q_1, s_1) \xrightarrow{a_{i_2}} (q_2, s_2) \xrightarrow{a_{i_3}} \cdots (q_{m-1}, s_{m-1}) \xrightarrow{a_{i_m}} (q', s')$ for some configurations $(q_1, s_1), \ldots, (q_{m-1}, s_{m-1})$ where $a_{i_1} a_{i_2} a_{i_3} \cdots a_{i_m} \in \mathcal{L}$. We write $(q, s) \xrightarrow{\mathcal{L}} (q', s')$ to mean this. The set of *reachable configurations* of $\mathcal{A}$ is given by:

$$\boldsymbol{R}(\mathcal{A}) := \{\ (q, s)\ :\ (q_0, \bot_n) \xrightarrow{\Sigma^*} (q, s)\ \}$$

The *configuration graph* $\mathcal{G}(\mathcal{A})$ of $\mathcal{A}$ has domain $\boldsymbol{R(\mathcal{A})}$, unary predicates Π and directed edges Σ between configurations. The ϵ-closure of such a graph $\mathcal{G}^{\epsilon}(\mathcal{A})$ is induced from $\mathcal{G}(\mathcal{A})$ by defining a-labelled edges between vertices related by $\xrightarrow{\epsilon^* a}$ in $\mathcal{G}(\mathcal{A})$.

2.4 Logics

We consider first-order logic $\boldsymbol{FO}$ on graphs as it is standardly defined. A $\Sigma_0 = \Pi_0 = \Delta_0$ formula $\phi(x_1, \ldots, x_k)$ is one without any quantifiers. A Σ_{n+1} formula is one of the form $\exists \vec{y}.\phi(\vec{y}, x_1, \ldots, x_k)$ where $\phi(\vec{y}, x_1, \ldots, x_k)$ is Π_n and a Π_{n+1} formula is one of the form $\forall \vec{y}.\phi(\vec{y}, x_1, \ldots, x_k)$ where ϕ is Σ_n. A Δ_n formula is one that is equivalent to both a Σ_n and Π_n formula on every CPDA graph. MSO is $\boldsymbol{FO}$ extended with variables ranging over sets. Transitive closure logic $\boldsymbol{FO(TC)}$ is $\boldsymbol{FO}$ together with a binary predicate $\overline{\phi(x,y)}$ for every formula of $\boldsymbol{FO}$ $\phi(x, y)$ with two variables, which denotes the transitive closure of the relation defined by $\phi(x, y)$. When the formulae transitively closed are restricted to Δ_1 we call the logic $\boldsymbol{FO(TC[\Delta_1])}$. A *sentence* is a formula with no free variables.

Note that we often allow ourselves to assume additional edges in the graph when writing formulae. For example $z \xrightarrow{pop_3;pop_2} z'$ means that we perform pop_3 and then pop_2 on the stack of z and move into a fixed distinguished control-state to result in a configuration z'. Thus $x \xrightarrow{pop_3;pop_2} z \ \wedge \ y \xrightarrow{pop_3;pop_2} z$ mean that pop_3 followed by pop_2 on the stack of x results in the same stack as pop_3 followed by pop_2 on the stack of y, regardless of whether x and y have different control-states. Provided that the number of operations constituting the edge is bounded, this can be done without ϵ-closure.

3 Undecidability

3.1 Post's Correspondence Problem

All of the undecidability results go via a reduction from the Post Correspondence Problem [16], which is well known to be undecidable. Consider a finite-alphabet Σ with $|\Sigma| \geq 2$. An instance of the Post Correspondence Problem (PCP) consists of two finite sequences of strings over Σ: $u_1, \ldots, u_m$ and $v_1, \ldots, v_m$. The question to be decided is whether there is a finite sequence $i_1 \ldots i_k$ consisting of integers $1 \leq i_j \leq m$ such that $u_{i_1}.u_{i_2}.\cdots.u_{i_k} = v_{i_1}.v_{i_2}.\cdots.v_{i_k}$.

▶ **Example 1.** Consider the following two sequences of strings over the alphabet $\{a, b, c\}$:

$$u_1 := ab \quad u_2 := cababcabb \quad u_3 := ca \qquad v_1 := ababc \quad v_2 := ab \quad v_3 := bca$$

Then the Post Correspondence Problem has answer 'yes' as witnessed by the solution 1123:

$$u_1.u_1.u_2.u_3 = ababcababcabbca = v_1.v_1.v_2.v_3$$

Given an instance of the PCP P we define a pushdown automaton $\mathcal{A}_1^P$ that pushes elements of Σ onto the stack together with indices indicating a partition into the u_i and v_i.

▶ **Definition 2.** The automaton $\mathcal{A}_1^P$ has stack alphabet: $\Sigma \uplus \{1_u, 2_u \ldots m_u\} \uplus \{1_v, 2_v \ldots m_v\}$ and behaves by non-deterministically choosing one of the following options:

- Push any member of Σ onto the stack.
- If the Σ symbols in the stack since the last symbol of the form i_u (or the bottom of the stack if there is no such symbol) form the word u_j, then it may push j_u onto the stack.
- If the Σ symbols in the stack since the last symbol of the form i_v (or the bottom of the stack if there is no such symbol) form the word v_j, then it may push j_v onto the stack.

P has a solution just in case $\mathcal{A}_1^P$ can generate a stack with 'matching' i_u and i_v subsequences.

▶ **Lemma 3.** *Let P be an instance of Post's Correspondence Problem. P has a solution just in case the automaton $\mathcal{A}_1^P$ can generate a stack s such that:*

- *$s_u = s_v$ where s_u is the subsequence of s consisting of elements of the form i_u and s_v of elements of the form i_v and equality is interpreted with respect to the indices i only.*
- *The top two elements of s form the set $\{i_u, i_v\}$ for some $1 \leq i \leq m$.*

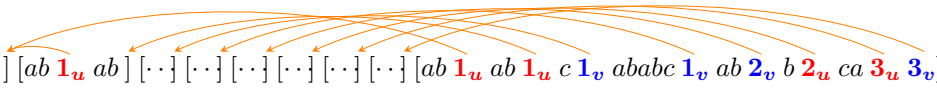
▶ **Example 4.** To continue the running example from Example 1, which we call P, the solution as represented by a stack of $\mathcal{A}_1^P$ is: $[\, ab\mathbf{1_u}ab\mathbf{1_u}c\mathbf{1_v}ababc\mathbf{1_v}ab\mathbf{2_v}b\mathbf{2_u}ca\mathbf{3_u}\mathbf{3_v}\,]$

3.2 Post's Correspondence Problem and 2-CPDA

Hague *et al.* [7] showed that the model-checking problem for MSO on 2-CPDA graphs is undecidable; indeed the 2-CPDA graph they exhibit witnesses the undecidability of transitive closure logic $\boldsymbol{FO(TC)}$. To introduce our basic technique, we first reprove the undecidability of $\boldsymbol{FO(TC)}$ on 2-CPDA graphs—in fact we get the undecidability of $\boldsymbol{FO(TC[\Delta_1])}$.

The 2-CPDA $\mathcal{A}_2^P$ is like $\mathcal{A}_1^P$ except it ensures each index (elements of the form i_u or i_v) has a pointer to a distinct 1-stack in the 2-stack. This enables first-order logic to 'ascertain corresponding positions' in two instances of a 1-stack by comparing the results of collapsing. More specifically, $\mathcal{A}_2^P$ behaves in the same way as $\mathcal{A}_1^P$ except that it performs $push_2; push_1^{j_u,2}$ or $push_2; push_1^{j_v,2}$ whenever $\mathcal{A}_1^P$ would have performed $push_1^{j_u}$ or $push_1^{j_v}$ for some $1 \leq j \leq m$.

▶ **Example 5.** To continue Example 1, the solution as represented by a stack of $\mathcal{A}_2^P$ is:



We adapt $\mathcal{A}_2^P$ further so that it may non-deterministically enter a distinguished control-state $\boldsymbol{guess}$ when it has formed a stack whose top two elements are those in a set $\{i_u, i_v\}$ for some $1 \leq i \leq m$. Lemma 3 then implies:

▶ **Lemma 6.** *Let P be an instance of Post's Correspondence Problem. P has a solution iff the automaton $\mathcal{A}_2^P$ can reach a configuration $(\boldsymbol{guess}, s)$ such that $s_u = s_v$, where s_u is the subsequence of $top_2(s)$ consisting of elements of the form i_u, and s_v of elements of the form i_v, and equality is interpreted with respect to the indices i only.*

Let us say that a stack s is a u-stack if its top element is of the form i_u and a v-stack if its top element is of the form i_v. Consider the compound operations $\boldsymbol{pop}_u(s)$ (resp. $\boldsymbol{pop}_v(s)$) that perform pop_1 on s, stopping only when the next element of the form i_u (resp. i_v) is found or else the top 1-stack is rendered empty if no such element exists. It is also useful to define $!u := v$ and $!v := u$. Note that a configuration $(\boldsymbol{guess}, s)$, where s is a w-stack for $w \in \{u, v\}$ (so that $pop_1(s)$ is a $!w$-stack), meets the criterion of Lemma 6 precisely when $top_1(\boldsymbol{pop}_w^k(s)) = i_w$ iff $top_1(\boldsymbol{pop}_{!w}^k(pop_1(s))) = i_{!w}$ for every $k \in \mathbb{N}$. This is because $\boldsymbol{pop}_u$ and $\boldsymbol{pop}_v$ step (backwards) through the sequences s_u and s_v.

Bearing this in mind, we extend $\mathcal{A}_2^P$ to an automaton $\mathcal{A}_{2+}^P$ that allows additional behaviour after reaching a $\boldsymbol{guess}$-configuration so that it does its best to verify the condition above. We will then use $\boldsymbol{FO(TC[\Delta_1])}$ to assert the constraints that the automaton is unable to enforce by itself. $\mathcal{A}_{2+}^P$ is defined to generate any sequence of configurations of the form:

$$(\boldsymbol{guess}, s) \xrightarrow{\boldsymbol{init}} (\boldsymbol{test}, t_1) \xrightarrow{\boldsymbol{next}} (\boldsymbol{test}, t_2) \xrightarrow{\boldsymbol{next}} \cdots \xrightarrow{\boldsymbol{next}} (\boldsymbol{test}, t_{k-1}) \xrightarrow{\boldsymbol{next}} (\boldsymbol{final}, t_k)$$

where the top three 1-stacks of each t_i include a 1-stack $\boldsymbol{u}(t_i)$ that is either a u-stack or empty and a 1-stack $\boldsymbol{v}(t_i)$ that is either a v-stack or empty. The automaton will abort if ever $top_1(\boldsymbol{u}(t_i)) = i_u$ but $top_1(\boldsymbol{v}(t_i)) = j_v$ with $i \neq j$ and will enter a control-state $\boldsymbol{final}$ iff $\boldsymbol{u}(t_k)$ and $\boldsymbol{v}(t_k)$ are both empty. Let us say that t_i is *w-dominant* for $w \in \{u, v\}$ if $\boldsymbol{w}(t_i) \sqsupset \boldsymbol{!w}(t_i)$, which occurs iff $\boldsymbol{w}(t_i)$ is below $\boldsymbol{!w}(t_i)$ in t_i. The third of the top three 1-stacks in t_i is the auxiliary *conditional stack* $\boldsymbol{cond}(t_i) = \boldsymbol{pop}_{!w}(\boldsymbol{!w}(t_i))$ where t_i is w-dominant. This will be of technical use in avoiding the need for ϵ-closure. Where s is a w-stack, the transition $\xrightarrow{\boldsymbol{init}}$ consists of just $push_2; pop_1; push_2; \boldsymbol{pop}_{!w}$, which sets up an initial $\boldsymbol{u}$, $\boldsymbol{v}$ and conditional stack, one of which will be the original $top_2(s)$.

The idea is that $\mathcal{A}^P_{2^+}$ can always enforce one of $\boldsymbol{u}(t_{i+1}) = \boldsymbol{pop}_u(\boldsymbol{u}(t_i))$ or $\boldsymbol{v}(t_{i+1}) = \boldsymbol{pop}_v(\boldsymbol{v}(t_i))$ but not both—the one that is not enforced in t_{i+1} must be guessed. It is easy to see that this is possible. A transition $(\boldsymbol{test}, t) \xrightarrow{\boldsymbol{next}} (\boldsymbol{test}, t')$ will employ a sequence of transitions of the form $pop_2; pop_2; pop_1^*; push_2; pop_1^*; push_2; pop_1^*$, enforcing $\boldsymbol{w}(t') = \boldsymbol{pop}_w(\boldsymbol{w}(t))$, guessing $\boldsymbol{!w}(t')$ but still enforcing $\boldsymbol{cond}(t') = \boldsymbol{pop}_{!w'}(\boldsymbol{!w'}(t'))$ when t is w-dominant and t' is w'-dominant. It may or may not be the case that $w = w'$ depending on the guess and the ordering of the i_u and i_v.

We now define a relation stronger than $\xrightarrow{\boldsymbol{next}}$ that $\mathcal{A}^P_{2^+}$ cannot impose by itself. We say that $c' = (\boldsymbol{test}, t')$ is a *successor* of $c = (\boldsymbol{test}, t)$, written $c' = c^+$, iff $c \xrightarrow{\boldsymbol{test}} c'$ and $\boldsymbol{!w}(t') = \boldsymbol{pop}_{!w}(\boldsymbol{w!}(t)) = \boldsymbol{cond}(t)$ where t is w-dominant. We then reformulate Lemma 6:

▶ **Lemma 7.** *Let $(\boldsymbol{guess}, s)$ be a reachable configuration of $\mathcal{A}^P_{2^+}$ for some instance P of Post's Correspondence Problem. Then s represents a solution to P in the sense of Lemma 6 if and only if there exists a chain of configurations $c_1, c_2, \ldots, c_k$ such that $c_{i+1} = c_i^+$ for $1 \leq i < k$, $(\boldsymbol{guess}, s) \xrightarrow{\boldsymbol{init}} c_1$ and c_k has control-state $\boldsymbol{final}$.*

Let us now observe that the relation $x = y^+$ is definable by a Δ_1 formula. Let w range over $\{u, v\}$ and consider configurations $(\boldsymbol{test}, t)$ and $(\boldsymbol{test}, t')$. Define $\boldsymbol{to_w}(t)$ to be $pop_2(t)$ or $(pop_2; pop_2)(t)$ as necessary to render $\boldsymbol{w}(t)$ the topmost 1-stack in t. Since the number of stack operations are bounded, we may safely view this as an edge in $\mathcal{G}(\mathcal{A}^P_{2^+})$ without resorting to ϵ-closure.

Recall that we have ensured that each instance of i_u or i_v in the representation of the postulated solution to P has a 2-link with a distinct target. It is also the case that the conditional stack of any t is the top-most 1-stack. This means that the equality $\boldsymbol{cond}(t) = \boldsymbol{w}(t')$ between *internal* 1*-stacks* holds iff the equality $collapse(t) = (\boldsymbol{to_w}; collapse)(t')$ holds, which may be viewed as an equality between *configurations* since t and t' are 2-stacks. Hence $x = y^+$ can be expressed by a Δ_1 formula $\psi_+(x, y) \wedge (x \xrightarrow{\boldsymbol{next}} y)$ where $\psi_+(x, y)$ asserts for each $w \in \{u, v\}$ that if y is $!w$-dominant, then:

$$\exists z.(x \xrightarrow{collapse} z \wedge y \xrightarrow{\boldsymbol{to_w}; collapse} z) \text{ equivalently } \forall z.(x \xrightarrow{collapse} z \rightarrow y \xrightarrow{\boldsymbol{to_w}; collapse} z)$$

With the aid of transitive closure, the existence of the $(\_)^+$-sequence in the condition in Lemma 7 can be asserted in Σ_1-$\boldsymbol{FO(TC[\Delta_1])}$, completing the reduction.

▶ **Lemma 8.** *There exists a Σ_1 sentence ϕ of $\boldsymbol{FO(TC[\Delta_1])}$ such that for all instances P of Post's Correspondence Problem we have: $\mathcal{G}(\mathcal{A}^P_{2^+}) \models \phi$ iff P has a solution.*

3.3 Undecidability for 3_2-CPDA and 4_2-CPDA

A 3_2-CPDA can record the chain of 2-stacks mentioned in Lemma 7 directly in its 3-stack—the members of the chain are piled on top of each other. This allows ϵ-closure in the graph and

universal quantification to replace transitive closure in the logic. As with $\mathcal{A}^P_{2^+}$ the key idea in the proof is to use first-order logic to assert the coherence of guesses that the automaton makes, ensuring that the 2-stacks making up the alleged $(\_)^+$-chain stored in the stack really do form a chain. The 3_2-CPDA $\mathcal{A}^P_{3_2}$ begins by behaving in the same way as $\mathcal{A}^P_{2^+}$, performing only 2-stack operations until just after it has performed an $\boldsymbol{init}$ transition. It then proceeds to perform $\boldsymbol{next}$ transitions, perfoming a $push_3$ operation after completing each one. It may stop and enter a distinguished control-state $\boldsymbol{guess}_{3_2}$ as soon as it has completed a $(push_3; \boldsymbol{next})$-transition that would have resulted in a $\boldsymbol{final}$-control state in $\mathcal{A}^P_{2^+}$. We thus have a solution to P iff $\mathcal{A}^P_{3_2}$ has a reachable configuration of the form:

$$(\boldsymbol{guess}_{3_2}, [s\ s_1\ s_2\ \cdots\ s_k]) \text{ such that } s_{i+1} = s_i^+ \text{ for every } 1 \leq i < k$$

Since the $s_i \xrightarrow{\boldsymbol{next}} s_{i+1}$ is guaranteed by the definition of $\mathcal{A}^P_{3_2}$, it suffices to assert that $\psi_+(s_i, s_{i+1})$ for every $1 \leq i \leq k$ (abusing notation by ignoring control-states). But observe how ψ_+ makes sense on 3-stacks provided that the 3-stacks only differ in their topmost 2-stacks. This is thus equivalent to $\psi_+([s\ s_1\ s_2\ \cdots\ s_{i-1}\ s_i\ s_i], [s\ s_1\ s_2\ \cdots\ s_{i-1}\ s_i\ s_{i+1}])$. Given ϵ-closure we are able to have a single edge labelled $\boldsymbol{toPre}$ going from a configuration of the form $(\boldsymbol{guess}_{3_2}, S)$ to any one of the form $(\boldsymbol{test}_{3_2}, S_i)$ where S_i is the prefix of S with its s_i on top for $2 \leq i \leq k$. Thus the following asserts correctness:

$$\exists x.(\boldsymbol{guess}_{3_2}(x) \ \wedge\ \forall y.(x \xrightarrow{\boldsymbol{toPre}} y \ \rightarrow\ \psi_+((pop_3; push_3)(y), y)))$$

▶ **Lemma 9.** *There exists a Σ_2 sentence ϕ of $\boldsymbol{FO}$ such that for all instances P of Post's Correspondence Problem we have: $\mathcal{G}^\epsilon(\mathcal{A}^P_{3_2}) \vDash \phi$ iff P has a solution.*

Observe that the sentence asserting the correctness of a chain suggested by $\mathcal{A}^P_{3_2}$ essentially says that every S_i generated by $\boldsymbol{toPre}$ from $(\boldsymbol{guess}_{3_2}, S)$ satisfies $\theta(S_i) = \theta'(S_i)$ for some particular sequences of operations θ and θ'. With the aid of an order-4_2 stack we are able to go from a $(\boldsymbol{guess}_{3_2}, [S])$ configuration to create both a configuration of the form $(\boldsymbol{test}_\theta, [S\ \theta(S_k)\ \theta(S_{k-1})\ \cdots\ \theta(S_2)])$ and following an alternative path a configuration of the form $(\boldsymbol{test}_{\theta'}, [S\ \theta'(S_k)\ \theta'(S_{k-1})\ \cdots\ \theta'(S_2)])$. In other words we create two 4_2-stacks containing in corresponding positions the 3_2-stacks that have to be compared to verify the chain. Given a 4_2-CPDA $\mathcal{A}^P_{4_2}$ implementing this, we can assert the existence of a correct chain with a Σ_1-sentence asserting the existence of a $\boldsymbol{test}_\theta$-configuration and a $\boldsymbol{test}_{\theta'}$-configuration both with the same stack. Whilst ϵ-closure is not required here, *the restriction to reachable configurations* is necessary for the control-states $\boldsymbol{test}_\theta$ and $\boldsymbol{test}_{\theta'}$ to be significant.

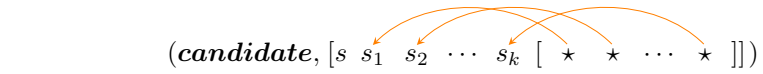
▶ **Lemma 10.** *There exists a Σ_1-sentence ϕ such that for every instance P of Post's Correspondence Problem we have $\mathcal{G}(\mathcal{A}^P_{4_2}) \vDash \phi$ iff P has a solution.*

3.4 The Non-Locality of 3_3-CPDA

Adapting $\mathcal{A}^P_{3_2}$ to become a 3_3-CPDA is straightforward—the role of 2-links, giving each element i_u and i_v a link with a distinct target, can easily be taken by 3-links; of course these distinct targets will be distinct 2-stacks rather than distinct 1-stacks. The challenge is that we also want to remove the need for ϵ-closure, although we still have to restrict the domain to configurations reachable from the origin. The idea is that 3-links are 'non-local' with respect to a 3-stack so that whilst with $\mathcal{A}^P_{3_2}$ it was necessary to use ϵ-closure in order to provide access to all elements of the alleged chain, this can instead, in some sense, be performed by a single *collapse* edge on a 3-link.

The 3_3-CPDA $\mathcal{A}^P_{3_3}$ begins by behaving in a similar way to $\mathcal{A}^P_{3_2}$ (using 3-links instead of 2-links). Recall a ***next***-transition in $\mathcal{A}^P_{2+}$ takes the form $pop_2; pop_2; pop_1^*; push_2; pop_1^*; push_2; pop_1^*$ (the lack of ϵ-closure is not a concern here as we will not need to refer to ***next***-transitions in the logic). By implication $\mathcal{A}^P_{3_2}$ will perform a ***next***-transition of the form: $push_3; pop_2; pop_2; pop_1^*; push_2; pop_1^*; push_2; pop_1^*$. We make $\mathcal{A}^P_{3_3}$ differ again from $\mathcal{A}^P_{3_2}$ by performing instead: $push_3; pop_2; pop_2; push_1^{\star,3}; push_2; pop_1^*; push_2; pop_1^*; push_2; pop_1^*$ for each ***next***-transition, where $\star$ is a new distinct stack symbol. After the ***next***-transitions are complete, and $\mathcal{A}^P_{3_2}$ would have moved into a $\boldsymbol{guess}_{3_2}$ control-state, we perform final $push_3; push_1^{\star,3}$ operations and move into a distinguished ***candidate*** control-state.

This has the effect of creating a configuration of the form:



which corresponds to an $\mathcal{A}^P_{3_2}$-configuration $(\boldsymbol{guess}_{3_2}, [s\ s_1\ s_2\ \cdots\ s_k\ ])$. From here we further extend $\mathcal{A}^P_{3_3}$ to be allowed to perform arbitarily long sequences of pop_2 operations from a ***candidate***-configuration with the option of ending in a distinguished control-state ***ready*** if it reaches a $\star$ symbol. Thus to assert the existence of a ***candidate***-configuration x such that $\chi(y)$ holds of all y that are prefixes of its stack ending in an s_i, we may say:

$$\exists x.\exists x'.\forall z.\forall y.(\boldsymbol{candidate}(x) \wedge x \xrightarrow{pop_3} x' \wedge ((\boldsymbol{ready}(z) \wedge z \xrightarrow{pop_3} x' \wedge z \xrightarrow{collapse} y) \rightarrow \chi(y)))$$

In particular we may take χ to be the assertion about prefixes used with $\mathcal{A}^P_{3_2}$ (noting that this does not require ϵ-closure). Again the restriction to reachable configurations is necessary to give ***candidate*** and ***ready*** their significance.

▶ **Lemma 11.** *Let P be an instance of Post's Correspondence Problem. Then there exists a Σ_2 sentence ϕ of **FO** such that: $\mathcal{G}(\mathcal{A}^P_{3_3}) \models \phi$ (note* no *ϵ-closure) if and only if P has a solution.*

As a consequence of all these reductions:

▶ **Theorem 12.** ***1.*** *For every $n \geq 4$ and $2 \leq m \leq n-2$ the Σ_1-**FO** model-checking problem for n_m-CPDA graphs (even without ϵ-closure, albeit restricted to configurations reachable from the origin) is undecidable.*

2. *For every $n \geq 3$ and $m \geq 3$ the Σ_2-**FO** model-checking problem for n_m-CPDA graphs (even w/o ϵ-closure, albeit restricted to configurations reachable from the origin) is undecidable.*

3. *For every $n \geq 3$ and $m \geq 2$ the Σ_2-**FO** model-checking problem for the ϵ-closures of n_m-CPDA graphs is undecidable.*

4 Σ_1 Decidability on n_n-CPDA

We reduce checking a Σ_1-sentence on the ϵ-closure of an n_n-CPDA graph to MSO model-checking on n-PDA, which is known to be decidable. The idea is to 'eliminate the n-links' and instead 'simulate' them in an n-PDA. The high level idea could be viewed as the graph analogue of Aehlig *et al.*'s link-elimination at order-2 for word generation [8]; however our technique for graphs is necessarily quite different. When generating a word one only needs to (non-deterministically) simulate a single run at a time, whilst for graphs one sometimes needs to verify the *non*-existence of edges which in some sense requires checking *all possible* runs.

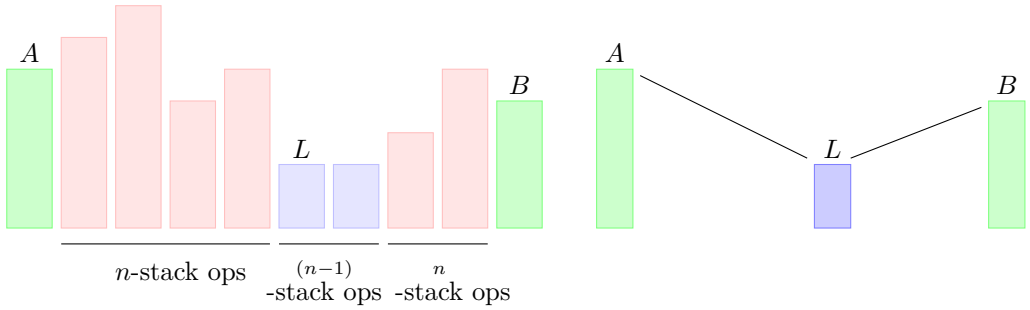


Figure 1 The anatomy of a run and eliminating the destruction of $(n-1)$-stacks.

Another issue is that for graphs links not only serve an operational purpose but additionally distinguish otherwise identical stacks. Due to space constraints we will not detail the solution to this challenge as it is largely technical. But the basic idea is that every atomic element (0-stack) and every constituent k-stack is assigned one of three colours $c_<$, $c_=$ and $c_>$ which specifies whether the highest target of an n-link that it contains is below, the same as or above the highest target of an n-link contained in a k-stack below it in the $(k+1)$-stack. The colouring of every prefix of a stack is sufficient to distinguish stacks even when links are deleted. It is also possible for an n_n-CPDA to keep track of sufficient information to dynamically maintain colour annotations.

We instead focus on the more interesting question of accounting for the operational aspect of links after they have been eliminated—that is how we express what a run involving *collapse* 'would have done' in the simulating n-PDA. Again due to space constraints we only sketch the main ideas. Our strategy is illustrated in Figure 1. Consider an a-labelled edge in the ϵ-closure of the configuration graph of an n_n-CPDA $\mathcal{A}$ between a configuration with stack A and a configuration with stack B, witnessed by an ϵ^*a-labelled run. Such a run can be divided into two parts with a configuration bearing stack L at the boundary. Suppose that no n-stack containing fewer than m $(n-1)$-stacks is used, then L is deemed to be the first configuration in the run with a stack containing m $(n-1)$-stacks.

We describe the ϵ^*-labelled first part of the run from A to L as an ϵ^*-*fall* and the ϵ^*a-labelled component from L to B as an ϵ^*a-*climb*. These two components are simulated by an n-PDA using two different methods. First consider the climb. Due to L being the 'lowest stack' in the run, it is possible to construct the stack of B from L without destroying any $(n-1)$-stacks and in particular without having to perform *collapse*. Of course the original n-CPDA might have a more complex run between L and B that does need to use *collapse*, but we show that it is possible to adjust the automaton so that it 'can smooth out' its run during the ϵ^*a-climb and avoid destroying any $(n-1)$-stacks. This means that climbs can be implemented *directly* by an n-PDA.

For the fall we exploit the fact that the stack of L is a prefix of the stack of A. The idea is that each $(n-1)$-stack t in an n-stack s will be annotated with information about the climbs that could be performed from the prefix of s ending in t. We then adjust the automaton further so that it is always aware of those annotations to which it has an ϵ^*-fall. This means that it can exploit the information in the annotations *indirectly* without actually having to 'physically' perform the ϵ^*-fall. Again this means that physical use of the *collapse* operation can be avoided.

In order to keep the annotations concerning climbs finite, it is necessary to fix in advance

a finite number of stacks to which one might eventually want to climb. These stacks will be possible witnesses to our Σ_1-sentence and so there will be one for each variable therein; the requirement to fix these is the reason why the proof does not generalise to more complex sentences. It is also impossible for an n-PDA to correctly provide such annotations itself and so, as with the undecidability proofs, it is necessary for it to guess these annotations and then use MSO to verify correctness externally.

So let us fix an n_n-CPDA $\mathcal{A}$ and a Σ_1-sentence $\exists x_1 \exists x_2 \cdots \exists x_k . \chi(x_1, x_2, \ldots, x_k)$ to be interpreted over $\mathcal{G}^{\epsilon}(\mathcal{A})$ where $\chi(x_1, x_2, \ldots, x_k)$ is quantifier free.

4.1 Simulating the Climb

An $\epsilon^* a$-climb is a run of the form: $(q, [s_1\ s_2\ \cdots\ s_k\ t]) \xrightarrow{\epsilon^* a} (q', [s_1\ s_2\ \cdots\ s_k\ t'\ t_1\ t_2\ \cdots\ t_m])$ that *never descends below* t so that for all stacks s occurring in the run, $[s_1\ s_2\ \cdots\ s_k] \sqsubset s$. Our first step is to construct a modified *'monotonic'* automaton $\mathcal{A}^{\uparrow}$ that can climb from the configuration transition to the last *without ever destroying any* $(n-1)$ *stacks* in the process.

We are able to construct $\mathcal{A}^{\uparrow}$ using *logical reflection* [2]. Given μ-calculus sentences $\phi_1, \ldots, \phi_k$, logical reflection allows a CPDA to be augmented so that it 'knows' (by reference to its control-state and the top element of the stack) whether a particular ϕ_i holds in its graph at its current configuration. In particular we can take the ϕ_i to be assertions of the form: 'If I were to to mark the current top_n $(n-1)$-stack and then perform a $push_n$ into control-state q, then I would be able to perform ϵ-transitions and end back at the marker in control-state q''. Given an automaton that is aware of the truth of such assertions, it can be adapted so that instead of actually performing these ϵ-transitions, which grow the stack before returning to the starting point, it can simply transition from control-state q to control-state q' without touching the stack. In this way it is able to carry out the climb without creating an $(n-1)$-stack only to destroy it later on, which is exactly what we want.

The CPDA produced by an application of logical reflection generates a graph isomorphic to the original in a particularly strong way. That is if a CPDA $\mathcal{B}'$ is produced from a CPDA $\mathcal{B}$ by an application of logical reflection, then there is an isomorphism $L : \mathcal{G}(\mathcal{B}) \cong (\mathcal{B}')$ that can be defined as a map on configurations of the form $L(q, s) = (q, L(s))$ where the control-state is preserved and L preserves the structure of the stack (albeit adding information to the atomic elements therein). Even though the evaluation of a μ-calculus formula will be a function of both q and $top_1(L(s))$ it is $top_1(L(s))$ that contains all of the additional information provided by logical reflection. (In actual fact this is a slight abuse of notation as $\mathcal{B}'$ will need to store information concerning the stack s in its control-state as well. However, since this information depends completely on the stack and not the control-state, it is safe for our purposes to suppress it notationally and assume that it is 'absorbed' into the $L(s)$ component). This means that when $\mathcal{A}^{\uparrow}$ transitions from control-state q to control-state q' without touching the stack, the information that the stack contains remains correct.

Using a similar technique we also allow $\mathcal{A}^{\uparrow}$ to construct all of its reachable configurations from its initial state without destroying $(n-1)$-stacks via transitions given a fresh label r.

4.2 Meta-Annotations—Towards simulating the Fall

Let Σ be the set of non-ϵ edge labels of $\mathcal{A}$ and let Q be its set of control-states, both of which are shared by $\mathcal{A}^{\uparrow}$. A *k-meta-annotation* (where k is the number of variables in the formula) is a $k.|\Sigma|$-tuple $M := ((Q_1^a)_{a\in\Sigma}, (Q_2^a)_{a\in\Sigma}, \ldots, (Q_k^a)_{a\in\Sigma})$ where $Q_i^a \subseteq Q$ for every i and a with $1 \leq i \leq k$ and $a \in \Sigma$. We further adapt $\mathcal{A}^{\uparrow}$ to non-deterministically choose some

meta-annotation M to push onto the stack before performing any $push_n$ operation. Hence every $(n-1)$-stack in any reachable configuration is decorated with a meta-annotation.

The '*meaning*' of meta-annotations is given with respect to a k-tuple of $\mathcal{A}^{\uparrow}$ configurations $c_1 = (q_1, s_1), c_2 = (q_2, s_2), \ldots, c_k = (q_k, s_k)$. Suppose that $t \sqsubseteq s_i$ is a prefix of one of the stacks of these configurations, then a 'correct' meta-annotation on top of t is one such that its set Q^a_j contains a control-state p iff there is an $\epsilon^* a$-climb from (p, t) to (q_j, s_j). Note in particular that if $pop_n(t) \not\sqsubseteq s_j$ then such a climb would be impossible and so $Q^a_j = \emptyset$.

If the meta-annotations contained in the stacks of $c_1, c_2, \ldots, c_k$ are all correct in this sense, then we say that this k-tuple of configurations is *consistent*. Consistency is not something that can be guaranteed by the automaton itself, but will be asserted using MSO.

We now consider how to handle ϵ^*-falls. Formally an ϵ^*-fall is a run of the form: $(q, [s_1\ s_2\ \cdots\ s_k]) \xrightarrow{\epsilon^*} (q', [s_1\ s_2\ \cdots\ s_{k'}])$ where $k' < k$ and $[s_1\ s_2\ \cdots\ s_{k'}]$ is a prefix of every stack occurring in the run.

We make one final adjustment to $\mathcal{A}^{\uparrow}$ by again applying logical reflection, this time with μ-calculus assertions of the form $\psi_{M,q}$ for each possible meta-annotation M and control-state q. The sentence $\psi_{M,q}$ says: 'From my current configuration I could behave as $\mathcal{A}$, without deploying any new meta-annotations, and eventually reach a stack with the meta-annotation M on top in control-state q'. Since no new meta-annotations are deployed, $\psi_{M,q}$ effectively says: 'I have an ϵ^*-fall from my current configuration to a configuration with control-state q and M on top of the stack'. Logical reflection gives the CPDA graph a unary predicate $\boldsymbol{M_q}^{\downarrow}$ that holds whenever $\psi_{M,q}$ would hold.

The *n-PDA* $\mathcal{A}^{\uparrow\downarrow}$ is formed from $\mathcal{A}^{\uparrow}$, adjusted as above, by removing all of the transitions performing a pop_n or *collapse* operation. Note that the $\boldsymbol{M_q}^{\downarrow}$ predicates depend only on the top_1 element of the stack and current control-state—these thus remain in $\mathcal{A}^{\uparrow\downarrow}$. Moreover since $\mathcal{A}^{\uparrow}$ was adapted to perform all $\epsilon^* a$-climbs without performing pop_n or *collapse*, then despite the deletion of edges, if $\mathcal{A}^{\uparrow}$ was able to perform an $\epsilon^* a$-climb from a configuration c to a configuration c', then $\mathcal{A}^{\uparrow\downarrow}$ can also perform such a climb. Hence consistency is preserved. The reachable configurations will also be unaffected by this deletion due to the r-transitions.

4.3 Σ_1-Model Checking

Any run of the form $(q, s) \xrightarrow{\epsilon^* a} (q', s')$, which witnesses an a-labelled edge in the ϵ-closure, is either a climb itself or else can be decomposed into $(q, s) \xrightarrow{\epsilon^*} (p, t) \xrightarrow{\epsilon^* a} (q', s')$ where the component from (q, s) to (p, t) is an ϵ^*-fall and the component from (p, t) to (q', s') is an $\epsilon^* a$-climb. We can thus assert the existence of an a-edge in $\mathcal{G}^{\epsilon}(\mathcal{A})$ by asserting in $\mathcal{G}(\mathcal{A})$ the existence of either a climb or else such a fall followed by a climb.

Given the quantifier-free first-order formula $\chi(x_1, x_2, \ldots, x_k)$, construct $\chi'(x_1, x_2, \ldots, x_k)$ by replacing every occurence of an atomic formula $x_i \xrightarrow{a} x_j$ with an MSO formula:

$$x_i \xrightarrow{\epsilon^* a} x_j \ \vee \bigvee_{M=((Q^a_1)_{a\in\Sigma},(Q^a_2)_{a\in\Sigma},\ldots,(Q^a_k)_{a\in\Sigma}),\ \ p\in Q^a_j} \boldsymbol{M_p}^{\downarrow}(x_i)$$

where the reachability assertion $x_i \xrightarrow{\epsilon^* a} x_j$ is MSO definable. Observe that in the n-PDA $\mathcal{A}^{\uparrow\downarrow}$ the relation $\xrightarrow{\epsilon^* a}$ asserts precisely the existence of an $\epsilon^* a$-climb and so if $c'_1, c'_2, \ldots, c'_k$ are *consistent* configurations of $\mathcal{A}^{\uparrow\downarrow}$ corresponding (via the isomorphism) to configurations $c_1, c_2, \ldots, c_k$ of $\mathcal{A}$, then $\mathcal{G}(\mathcal{A}^{\uparrow\downarrow}) \vDash \chi'(c'_1, c'_2, \ldots, c'_k)$ iff $\mathcal{G}^{\epsilon}(\mathcal{A}) \vDash \chi(c_1, c_2, \ldots, c_k)$. But note further that since for every k-tuple of configurations of $\mathcal{A}$ there exists *precisely one*

corresponding *consistent* k-tuple of configurations of $\mathcal{A}^{\uparrow\downarrow}$ we have:

$$\mathcal{G}(\mathcal{A}^{\uparrow\downarrow}) \models \exists x_1 x_2 \cdots x_k.(\textbf{\textit{con}}(x_1, x_2, \ldots, x_k) \wedge \chi'(x_1, x_2, \ldots, x_k)) \quad \text{iff } \mathcal{G}^{\epsilon}(\mathcal{A}) \models \exists x_1 x_2 \cdots x_k.\chi(x_1, x_2, \ldots, x_k)$$

where $\textbf{\textit{con}}(x_1, x_2, \ldots, x_k)$ asserts consistency. Note that $\textbf{\textit{con}}(x_1, x_2, \ldots, x_k)$ is also MSO definable in $\mathcal{G}(\mathcal{A}^{\uparrow\downarrow})$; one can quantify over prefixes of a stack by iterated pop_n, and reachability in $\mathcal{G}(\mathcal{A}^{\uparrow\downarrow})$ captures precisely the climbs in $\mathcal{A}$. This completes the reduction.

5 Further Directions

Whilst it is very natural to restrict to configurations reachable from the origin of the graph, it would also be interesting to investigate removing this restriction. For collapsible stacks there would be two versions of this problem: allowing arbitrary stack contents in configurations or restricting to stacks that could be constructed from the empty stack using stack operations. We conjecture that the former is undecidable but that the latter is decidable.

Acknowledgement We thank the anonymous reviewers for helpful comments.

References

1. C.H Broadbent. *On Collapsible Pushdown Automata, their Graphs and the Power of Links.* PhD thesis, 2011.
2. C.H. Broadbent, A. Carayol, C.-H.L. Ong, and O. Serre. Recursion Schemes and Logical Reflection. In *LICS*, 2010.
3. T. Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In *ICALP*, 2003.
4. A. Carayol. Regular Sets of Higher-Order Pushdown Stacks. In *MFCS*, 2005.
5. A. Carayol and S. Wöhrle. The Caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS*, 2003.
6. D. Caucal. On infinite transition graphs having a decidable monadic theory. *Theoretical Computer Science*, 290(1):79–115, 2003.
7. M. Hague, A.S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible Pushdown Automata and Recursion Schemes. In *LICS*, 2008.
8. K. Aehlig, J. G. de Miranda, and C.-H. L. Ong. Safety is not a restriction at level 2 for string languages. In *FoSSaCS*, 2005.
9. A. Kartzow. Collapsible Pushdown Graphs of Level 2 are Tree-Automatic. In *STACS*, 2010.
10. T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-Order Pushdown Trees are Easy. In *FoSSaCS*, 2002.
11. T. Knapik, D. Niwinski, P. Urzyczyn, and I. Walukiewicz. Unsafe Grammars and Panic Automata. In *ICALP*, 2005.
12. N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, 2009.
13. C.-H.L. Ong and S.J. Ramsay. Verifying Higher-Order Functional Programs with Pattern-Matching Algebraic Data Types. In *POPL*, 2011.
14. P. Parys. Collapse Operation Cannot Be Simulated Even By Using Higher Levels (Unpublished). 2011.
15. P. Parys. Collapse Operation Increases Expressive Power of Deterministic Higher Order Pushdown. In *STACS*, 2011.
16. E. Post. A variant of a recursively unsolvable problem. *Bulletin of the American Mathematical Society*, 52(4):264–268, 1946.

The Power of Local Search: Maximum Coverage over a Matroid

Yuval Filmus[1,2] **and Justin Ward**[1]

1 **Department of Computer Science, University of Toronto**
{yuvalf,jward}@cs.toronto.edu
2 **Supported by NSERC**

Abstract

We present an optimal, combinatorial $1 - 1/e$ approximation algorithm for Maximum Coverage over a matroid constraint, using non-oblivious local search. Calinescu, Chekuri, Pál and Vondrák have given an optimal $1-1/e$ approximation algorithm for the more general problem of monotone submodular maximization over a matroid constraint. The advantage of our algorithm is that it is entirely combinatorial, and in many circumstances also faster, as well as conceptually simpler.

Following previous work on satisfiability problems by Alimonti, as well as by Khanna, Motwani, Sudan and Vazirani, our local search algorithm is *non-oblivious*. That is, our algorithm uses an auxiliary linear objective function to evaluate solutions. This function gives more weight to elements covered multiple times. We show that the locality ratio of the resulting local search procedure is at least $1 - 1/e$. Our local search procedure only considers improvements of size 1. In contrast, we show that oblivious local search, guided only by the problem's objective function, achieves an approximation ratio of only $(n - 1)/(2n - 1 - k)$ when improvements of size k are considered.

In general, our local search algorithm could take an exponential amount of time to converge to an *exact* local optimum. We address this situation by using a combination of *approximate* local search and the same partial enumeration techniques as Calinescu et al., resulting in a clean $(1 - 1/e)$-approximation algorithm running in polynomial time.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases approximation algorithms; maximum coverage; matroids; local search

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.601

1 Introduction

Maximum coverage [1,6–9,11,13,16] (also known as Max Cover) is a well-known combinatorial optimization problem related to Set Cover. Given a universe U, element weights $w\colon U \to \mathbb{R}_+$, a family $\mathcal{F} \subset 2^{2^U}$ of subsets of U, and a number n, the problem is to select n sets $S_i \in \mathcal{F}$ such that $w(S_1 \cup \cdots \cup S_n)$ is as large as possible.

Like many combinatorial optimization problems, maximum coverage is hard to solve exactly. A straightforward reduction from Set Cover shows that the decision version of maximum coverage with unit weights (deciding whether there are n sets that span at least m elements) is NP-complete, so the best we can hope for is an approximation algorithm.

One natural approach is the following heuristic. First pick the set S_1 of maximum weight. Then pick the set S_2 that maximizes $w(S_2 \setminus S_1)$, and so on. This approach leads to the well-known *greedy algorithm*, and yields an approximation ratio of $1 - 1/e \approx 0.632+$. Amazingly, in this setting the greedy algorithm is optimal. Feige [9] used the PCP theorem to

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 601–612

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


show that if there exists a polynomial-time algorithm that approximates maximum coverage within a ratio of $1 - 1/e + \epsilon$ for some $\epsilon > 0$ then $\mathsf{P} = \mathsf{NP}$.

The present paper deals with a generalized version of maximum coverage, in which we are given a matroid $\mathfrak{m}$ over $\mathcal{F}$, and the goal is to find a collection $S \subseteq \mathcal{F}$ that covers elements of maximum weight, subject to the constraint that $S \in \mathfrak{m}$. We call this problem *maximum coverage over a matroid constraint*. If the underlying matroid is the uniform matroid of rank n, we recover the original problem. Because every maximum coverage function is monotone submodular, this problem falls under the more general setting of maximizing monotone submodular functions subject to a matroid constraint. The greedy algorithm still applies in this general setting, but its approximation ratio is only $1/2$, even when the monotone submodular function is a maximum coverage function. Calinescu, Chekuri, Pál and Vondrák [5] developed a sophisticated algorithm for a general problem of monotone submodular maximization over a matroid constraint, which achieves the optimal approximation ratio of $1 - 1/e$. Their algorithm first finds a fractional solution using the *continuous greedy* algorithm, and then rounds it to an integral solution using *pipage rounding*.

1.1 Our contribution

We propose a simple algorithm for maximum coverage over a matroid constraint. Like the continuous greedy algorithm, our algorithm achieves the optimal approximation ratio of $1 - 1/e$. In contrast to the continuous greedy algorithm, however, our algorithm is *combinatorial*, in the sense that it only deals with integral solutions. Our approach is based on *non-oblivious local search*, a technique first proposed by Alimonti [2] and by Khanna, Motwani, Sudan and Vazirani [12].

In classical (or, *oblivious*) local search, the algorithm starts at an arbitrary solution, and proceeds by iteratively making small changes that improve the objective function, until no such improvement can be made. The *locality ratio* of a local search algorithm is $\min w(S)/w(O)$, where S is a solution that is locally-optimal with respect to the small changes considered by the algorithm, O is a global optimum, and w is the objective function for the problem. The locality ratio provides a natural, worst-case guarantee on the approximation performance of the local search algorithm.

In many cases, oblivious local search may have a very poor locality ratio, implying that a locally-optimal solution may be of significantly lower quality than the global optimum. For example, for our problem the locality ratio for an algorithm changing a single set at each step is $1/2$. As the locality ratio depends on the type and size of local changes considered, one approach for improving an algorithm's performance is simply to consider larger (but still constant-sized) neighborhoods. Unfortunately, in our case, this technique results in no asymptotic improvement in the locality ratio. Specifically, we show that the locality ratio of oblivious local search remains only $(n-1)/(2n-1-k)$ when k sets are exchanged. For constant k, this is only marginally better than the approximation ratio $1/2$ achievable using the greedy algorithm.

Non-oblivious local search adopts a more radical approach by altering the objective function used to guide the search. We proceed as before, but modify the local search algorithm to use an auxiliary objective function to decide whether the current solution is an improvement. By carefully choosing this auxiliary function, we can ensure that poor local optima with respect to the original objective function are no longer local optima.

In this paper, we present a non-oblivious local search algorithm for the problem of maximum coverage over a matroid constraint. Specifically, we construct an auxiliary objective

function whose locality ratio (for single changes) is slightly larger than $1 - 1/e$, resulting in an algorithm whose approximation ratio is the best possible, assuming $\mathsf{P} \neq \mathsf{NP}$.

In general, local search algorithms could converge in exponentially many steps. We address this issue using approximate local search, a technique described systematically by Schulz, Orlin and Punnen [15]. Approximate local search allows us to bound the running time of the algorithm at a small cost in the resulting approximation ratio. By employing partial enumeration, as described by Calinescu et al., we can eliminate this small cost, achieving an approximation ratio of $1 - 1/e$.

1.2 Comparison with existing algorithms

Both our algorithm and the one by Calinescu et al. give the same approximation ratio. The continuous greedy algorithm works on a discretized version of a particular continuous relaxation of the problem to obtain a fractional solution to the problem. This solution must then be rounded to an integral solution using the pipage rounding technique, which employs a submodular minimization algorithm at each step. In contrast, our algorithm always maintains a current integral solution and requires only simple combinatorial operations that add and remove elements from this solution. Moreover, our algorithm is extremely simple, and can be described using only a few lines of pseudocode.

In most settings, our algorithm is also faster. For a fair comparison, we consider versions of both algorithms that achieve an approximation ratio of at least $1 - 1/e$. Furthermore, we analyze the algorithm of Calinescu et al. only in the special case of maximum coverage. In this setting, it is possible to calculate the continuous relaxation of the objective function exactly, rather than by random sampling, thus greatly improving the runtime of the continuous greedy algorithm. Denoting the rank of the matroid by n, the total number of sets by $s = |\mathcal{F}|$, the maximal size of a set by u, and the sum of the sizes of all sets by U, our algorithm runs in time $\tilde{O}(n^3 s^2 u)$, whereas the algorithm by Calinescu et al. runs in time $\tilde{O}(n^2 s^3 u + s^7)$. For all non-trivial instances of the problem, we must have $s > n$, and so our algorithm is indeed considerably faster, assuming that the size u of the largest set does not dominate both expressions. We stress that this is the case even after the continuous greedy algorithm has been optimized to compute the maximum coverage function directly, rather than by sampling as in the general analysis of Calinescu et al. [5].

Another relevant algorithm is described in an earlier paper by the same set of authors [4]. The algorithm presented by Calinescu et al. in that paper is more general than ours, but less general than their later paper. It applies to monotone submodular functions which are sums of weighted rank functions of matroids. The continuous greedy algorithm is replaced with a simple linear program. In the simplest case of a uniform matroid, the running time of this algorithm (using interior-point methods) is $\tilde{O}(U^{3.5} + s^7)$; more complicated matroids result in even worse running times. When the sets are large, this is considerably slower than our algorithm.

1.3 Future work

We believe that the approach outlined in this paper can be used to design approximation algorithms for similar problems. In particular, we are working on generalizing the algorithm to monotone submodular functions.

1.4 Paper organization

Section 2 provides a concise overview of existing work related to maximum coverage. In Section 3 we formally present the problem, discuss the limitations of oblivious local search, and present our non-oblivious local search algorithm. In Section 4 we give an analysis of the approximation ratio and runtime for our non-oblivious local search algorithm, and show how to attain a clean, polynomial time $(1 - 1/e)$-approximation by using partial enumeration techniques, as in [5].

For reasons of space, we defer some proofs to the appendix, which only appears in the full version of the paper that can be found on the authors' websites. In the appendix we also discuss the optimality of our auxiliary objective function, and analyze the standard oblivious local search in the special case of strongly base orderable matroids, showing that its performance can be much worse than our non-oblivious local search algorithm. Finally, we show that the difficulty of finding an exact local optimum is not an artifact of our oblivious non-oblivious objective function by presenting an instance for which even the oblivious local search algorithm requires an exponential number of improvements to find an exact local optimum.

1.5 Acknowledgments

We thank Anupam Gupta for suggesting the problem to us, and Allan Borodin and Roy Schwartz for helpful discussions.

2 Related work

Maximum coverage was first defined by Hochbaum and Pathria [11] in 1998. In fact, an even more general problem had been defined earlier by Cornuejols, Fisher and Nemhauser [8] in 1977, in the context of locating bank accounts. Both papers describe the greedy algorithm, and show that its approximation ratio is $1 - (1 - 1/n)^n \approx 1 - 1/e$.

Feige [9], in his seminal paper on the inapproximability of Set Cover, showed that unless $\mathsf{P} = \mathsf{NP}$, it is impossible to approximate maximum coverage to within $1 - 1/e + \epsilon$ for any $\epsilon > 0$. His proof uses PCP techniques, extending earlier work by Lund and Yannakakis [14].

Maximum coverage over a partition matroid was considered by Chekuri and Kumar [6] under the name *maximum coverage with group budget constraints*. In this variant, the family $\mathcal{F}$ is partitioned into subfamilies $\mathcal{F}_i$, and the solution must contain at most n_i sets from $\mathcal{F}_i$, and at most n overall. They analyze the performance of the greedy algorithm when the greedy choices are given by an approximate oracle.

Algorithms for maximum coverage with group budget constraints with the optimal approximation ratio $1-1/e$ were presented by Srinivasan [16] and by Ageev and Sviridenko [1]. Both algorithms first formulate the problem as an LP, and then round the solution. Srinivasan's approach involves sophisticated sampling. Ageev and Sviridenko's approach, *pipage rounding*, repeatedly simplifies the solution until it becomes integral, without decreasing its value. Both approaches work in more general settings.

Calinescu, Chekuri, Pál and Vondrák [5] combined a more sophisticated version of pipage rounding with the *continuous greedy* algorithm to obtain an optimal $1 - 1/e$ approximation algorithm for monotone submodular maximization over a matroid constraint. The continuous greedy algorithm is a steepest descent algorithm running in continuous time (in practice, a suitably discretized version is used), producing a fractional solution. This solution is rounded using pipage rounding.

Other generalizations of maximum coverage appear in the literature. In *budgeted maximum coverage*, each element is provided with a cost, and the sets chosen by the algorithm are restricted by the total cost of the elements they cover. Khuller, Moss and Naor [13] show that a greedy approach yields an optimal $1 - 1/e$ approximation algorithm. Cohen and Katzir [7] generalize this even further, and provide an optimal $1 - 1/e$ semi-greedy approximation algorithm.

3 Local search for maximum coverage over a matroid

We consider the problem of maximum coverage over a matroid. The inputs are

- A universe U.
- Value oracle access to a non-negative weight function $w\colon U \to \mathbb{R}_+$.
- A family $\mathcal{F}$ of subsets of U with $|\mathcal{F}| = s$, $\max_{A\in\mathcal{F}} |A| = u$.
- A matroid over $\mathcal{F}$ of rank n, given as an independence oracle.

Note that the matroid $\mathfrak{m}$ has as its ground set the *sets* $\mathcal{F}$, and so a member of $\mathfrak{m}$ is a collection of sets from $\mathcal{F}$. We call the members of $\mathfrak{m}$ *independent sets*. We extend w to a function over subsets A of U by letting $w(A) = \sum_{u\in A} w(u)$. The goal, then, is to find a collection of sets $S \subseteq \mathcal{F}$ that covers elements in U of maximal weight, subject to the constraint that $S \in \mathfrak{m}$:

$$\max_{S\in\mathfrak{m}} w(\bigcup S).$$

Recall that $\mathfrak{m}$ is a matroid over $\mathcal{F}$ if: (1) $\mathfrak{m} \neq \emptyset$, (2) $\mathfrak{m}$ is downward-closed (if $A \in \mathfrak{m}$ and $B \subset A$, then $B \in \mathfrak{m}$), and (3) for all $A, B \in \mathfrak{m}$ with $|A| > |B|$ we have $B \cup \{x\} \in \mathfrak{m}$ for some $x \in A \setminus B$. This last property guarantees that all maximal independent sets of the matroid have the same cardinality. These sets are called *bases*, and their common cardinality is called the *rank*, denoted in this paper by n.

Our starting point is the oblivious local search algorithm, shown in Algorithm 1. The algorithm starts from an arbitrary base S (obtained, e.g., by the standard greedy algorithm) and repeatedly attempts to improve the total weight of all elements covered by S by exchanging up to k sets in S with k sets not in S, maintaining the matroid constraint. We call a pair (A, B) a *k-exchange* for S if $A \subseteq S$ with $|A| \leq k$, $B \subseteq \mathcal{F} \setminus S$ with $|B| = |A|$. When no single k-exchange improves the weight of all elements covered by S, the algorithm returns S.

Algorithm 1 Oblivious k-LocalSearch

$S \leftarrow$ an arbitrary basis in $\mathfrak{m}$.
repeat
  $S_{\text{old}} \leftarrow S$ {Remember previous solution}
  Let $\mathcal{E}$ be the set of all valid k-exchanges for S
  $S \leftarrow \arg\max_{(A,B)\in\mathcal{E}} w\left(\bigcup (S \setminus A \cup B)\right)$
until $S = S_{\text{old}}$ {Repeat until no improvement is possible}
return S

As mentioned in the introduction, the locality ratio of Algorithm 1 is rather poor. Consider the following set system:

$$A_1 = \{x, \epsilon_A\}, \qquad A_2 = \{\epsilon_B\},$$
$$B_1 = \{y\}, \qquad B_2 = \{x\}.$$

The elements x, y have unit weight, and the elements ϵ_A, ϵ_B have some arbitrarily small weight $\epsilon > 0$. We consider the partition matroid whose independents sets contain at most one of $\{A_i, B_i\}$ for $i \in \{1, 2\}$. Under this matroid, the base $\{A_1, A_2\}$ is a local optimum for 1-exchanges, since replacing A_1 by B_1 or A_2 by B_2 both result in a net loss of ϵ. The global optimum is $\{B_1, B_2\}$, and the locality ratio is thus $(1 + 2\epsilon)/2$. Note that the base $\{A_1, A_2\}$ is also produced by the standard greedy algorithm applied to this instance. This shows that we cannot hope to beat the locality ratio by choosing a greedy starting solution.

We can generalize this example to show that oblivious k-local search has a locality ratio of at most $\frac{n-1}{2n-k-1}$ for all $k < n$. Let the universe U consist of $n-1$ elements $\{x_1, \ldots, x_{n-1}\}$ and $n - k$ elements $\{y_1, \ldots, y_{n-k}\}$, all of weight 1, and $n - 1$ elements $\{\epsilon_2, \ldots, \epsilon_n\}$ of arbitrarily small weight $\epsilon > 0$. For each $1 \leq i \leq n$, there are two sets A_i and B_i, defined as follows:

$$A_i = \{\epsilon_i\} \text{ for } 1 \leq i \leq n-1, \qquad A_n = \{x_1, \ldots, x_{n-1}\},$$
$$B_i = \{x_i\} \text{ for } 1 \leq i \leq n-1, \qquad B_n = \{y_1, \ldots, y_{n-k}\}.$$

We consider the partition matroid whose independent sets contain at most one of $\{A_i, B_i\}$ for each $i \in [n]$. The globally optimal solution is the set $B = \{B_i\}_{1 \leq i \leq n}$, which covers elements of total weight $n - 1 + n - k = 2n - k - 1$. The set $A = \{A_i\}_{1 \leq i \leq n}$ is locally optimal under improvements of size k and covers elements of total weight only $(n-1)(1+\epsilon)$, and the locality ratio is thus $(n-1)(1+\epsilon)/(2n-k-1)$. In order to see that it is, in fact, a local optimum, note that if we do not replace A_n with B_n, the remaining replacements can only decrease the value of the solution. Suppose, then, that we do replace A_n with B_n. This replacement reduces the total weight of the covered elements by $k - 1$. There are only $k - 1$ remaining exchanges, each of which can increase the total weight of the covered elements by less than 1. Again, we note that the solution A is also the solution produced by the greedy algorithm. In the special case of *strongly base orderable* matroids, the approximation ratio of oblivious k-local search is, in fact, exactly $\frac{n-1}{2n-k-1}$, as we show in the full version of the paper.

Let us examine the first instance above in more detail. Intuitively, the basis $\{A_1, B_2\}$ in our example is better than the basis $\{A_1, A_2\}$ since it is of almost equal value to $\{A_1, A_2\}$ but additionally covers element x twice. From a local search perspective, this is an advantage since it ensures that x will stay covered after the next exchange. In order to improve the performance of local search, we want to somehow give extra weight to solutions that offer flexibility in future exchanges, perhaps even at a slight loss in the objective function. Following this intuition, we employ a function which gives extra weight to elements that are covered multiple times as an auxiliary objective function. A similar idea appears in Khanna et al. [12], in the context of the maximum satisfiability problem, and even earlier in similar work by Alimonti [2]. There, the idea is to give extra weight to clauses which are satisfied by more than one variable, because these clauses will remain satisfied after flipping the next variable in the search procedure.

Thus, we seek to modify Algorithm 1 by replacing the oblivious objective function w with an auxiliary objective function f of the general form:

$$f(S) = \sum_{u \in U} \alpha_{h_u(S)} w(u),$$

where $h_u(S) = |\{A \in S : u \in A\}|$ is the number of sets in S that contain element u. By setting, for example $\alpha_i > \alpha_1$ for all $i > 1$, we can give extra value to solutions that cover some element more than once. Additionally, note that if we set $\alpha_0 = 0$ and $\alpha_i = 1$ for all $i > 0$, we recover the oblivious objective function. We now consider the problem of how to set the α_i. We want to balance two concerns. First, we want to allow the algorithm to potentially decrease the objective value of the current solution in the short-term, in exchange for future flexibility. However, in the long-term, we want the algorithm to give enough weight to the original objective that it produces a reasonable final solution.

There is no immediately compelling reason to assign any weight to elements that have not been covered at all, so let us set $\alpha_0 = 0$, and examine the above instance in terms of the remaining α_i. We have

$$f(\{A_1, A_2\}) = (1 + 2\epsilon)\alpha_1, \qquad f(\{A_1, B_2\}) = \alpha_2 + \epsilon\alpha_1,$$
$$f(\{B_1, A_2\}) = (1 + \epsilon)\alpha_1, \qquad f(\{B_1, B_2\}) = 2\alpha_1.$$

By setting $\epsilon = \frac{\alpha_2 - \alpha_1}{\alpha_1}$, the solution $\{A_1, A_2\}$ will remain a local optimum, even for the non-oblivious potential function. The locality ratio (in terms of the *original* objective function) will remain $1/2 + \epsilon = 1/2 + \frac{\alpha_2 - \alpha_1}{\alpha_1}$. If we set α_2 to be too close to α_1, there will not be much improvement in the locality ratio, and if $\alpha_2 < \alpha_1$, the locality ratio will *decrease*. This confirms our intuition that it is advantageous to give more weight to elements that are covered multiple times. Alternatively, if we set $\alpha_2 \geq 2\alpha_1$, then the solution $\{A_1, B_2\}$ will become a local optimum, and the locality ratio will become $1/2 + \epsilon/2$. This confirms our intuition that it is bad to give *too much* extra weight to elements covered multiple times.

By extending a similar analysis to arbitrary instances, we obtain the following values for α_i:

$$\alpha_0 = 0, \qquad \alpha_1 = 1 - \frac{1}{e^{[n]}}, \qquad \alpha_{i+1} = (i+1)\alpha_i - i\alpha_{i-1} - \frac{1}{e^{[n]}}, \tag{1}$$

where the constant $e^{[n]}$ is defined as

$$e^{[n]} = \sum_{l=0}^{n-1} \frac{1}{l!} + \frac{1}{(n-1)!(n-1)}.$$

This choice of α_i is optimal, as we show in the full version of the paper. We note that the sequence $e^{[i]}$, where $i \geq 2$, is decreasing and bounded below by e:

▶ **Lemma 1.** *For all $n \geq 2$ we have $e < e^{[n]}$ and $e^{[n]} > e^{[n+1]}$.*

Our coefficients α_i satisfy the following properties, which follow directly from their definition.

▶ **Lemma 2.** *Let $\beta_i = \alpha_{i+1} - \alpha_i$. Then, the β_i satisfy the recurrence relation*

$$\beta_0 = 1 - \frac{1}{e^{[n]}}, \quad \beta_i = i\beta_{i-1} - \frac{1}{e^{[n]}}.$$

▶ **Lemma 3.** *For all $i < n$, $\beta_i > 0$ and $\beta_{i+1} \leq \beta_i$.*

▶ **Lemma 4.** *There exists a universal constant C_0 such that for all $i \leq n$, $\alpha_i \leq C_0 \log i$.*

Our resulting local search algorithm for maximum coverage over a matroid is given in Algorithm 2. In addition to using the non-oblivious potential function f described above, we modify our algorithm to start from a greedy initial solution. This initial solution is a good starting point, and speeds up the convergence of the algorithm. Our algorithm takes as a parameter δ, which governs how much an improvement is required to improve the current solution to be accepted. We describe this aspect of the algorithm in more detail in the next section.

Algorithm 2 LocalSearch(δ)

{Greedy algorithm}
$S \leftarrow \varnothing$
for $i = 1 \to n$ **do**
  $S \leftarrow S + \underset{A \in \mathcal{F} : S+A \in \mathfrak{m}}{\arg\max} [f(S + A) - f(S)]$
end for
{Local search}
repeat
  $S_{\text{old}} = S$ {Remember previous solution}
  Let $\mathcal{E}$ be the set of all valid 1-exchanges for S
  $S \leftarrow \arg\max_{(A,B)\in\mathcal{E}} f((S \setminus A) \cup B)$
until $S \leq (1 + \delta)S_{\text{old}}$ {Repeat until δ-locally optimal}
return S_{old}

4 Analysis of the non-oblivious local search algorithm

Our analysis makes use of the notion of a *δ-approximate local optimum*. Formally, we say that a solution S is a δ-approximate local optimum if $(1+\delta)f(S) \geq f(S')$ for all solutions S' differing from S by a single set. Intuitively, replacing exact local optimality (as in Algorithm 1) with approximate local optimality limits the total number of improvements the algorithm can make, at a slight cost in the approximation factor. We consider some δ-approximate local optimum $S = \{S_1, \ldots, S_n\}$ and some global optimum $O = \{O_1, \ldots, O_n\}$. A classical result of Brualdi [3] shows that for any matroid $\mathfrak{m}$ we can renumber the sets of O so that for each i, $S_{-i}, O_i := (S \setminus \{S_i\}) \cup \{O_i\}$ is a base of $\mathfrak{m}$. We suppose that O has been renumbered so that this is the case, and consider the 1-exchanges that remove S_i from S and add O_i to the result. Local optimality implies that for each i, we have

$$(1 + \delta)f(S) \geq f(S_{-i}, O_i).$$

Summing over all n such inequalities we obtain the inequality

$$n(1 + \delta)f(S) \geq \sum_{i=1}^{n} f(S_{-i}, O_i). \tag{2}$$

The main difficulty of the analysis lies in the fact that inequality (2) is given in terms of the non-oblivious potential function f, but we wish to derive an approximation guarantee for the original objective function w. In order to put f and w on common ground, we make the following definitions.

For any two subsets $L, G \subset [n]$, we define $E_{L,G}$ to be the set of elements that belong to the sets S_i for $i \in L$, and O_j for $j \in G$, and no other sets in S and O. The sets $E_{L,G}$ thus form a partition of U. Then, for all integers $l, c, g \geq 0$ such that $1 \leq l + c + g \leq n$, we define

$$x_{l,c,g} = \sum_{\substack{L,G: \\ |L|=l+c, \\ |G|=g+c, \\ |L\cap G|=c}} w(E_{L,G}).$$

In words, $x_{l,c,g}$ is the total weight of elements that belong to exactly $l + c$ of the sets S_i, exactly $g + c$ of the sets O_j, exactly c of them sharing indices. We call the variables $x_{l,c,g}$ *symmetric variables*.

We can express all the quantities we are interested in using the symmetric variables $x_{l,g,c}$:

$$f(S) = \sum_{l+c\geq 1} \alpha_{l+c} x_{l,c,g} = \sum_{l,c,g} \alpha_{l+c} x_{l,c,g} \text{ (since } \alpha_0 = 0),$$

$$\sum_{i=1}^{n} f(S_{-i}, O_i) = \sum_{l,c,g} (l\alpha_{l+c-1} + g\alpha_{l+c+1} + (n-l-g)\alpha_{l+c}) x_{l,c,g},$$

$$w\left(\bigcup S\right) = \sum_{l+c\geq 1} x_{l,c,g},$$

$$w\left(\bigcup O\right) = \sum_{g+c\geq 1} x_{l,c,g}.$$

The only expression which is not immediate is the one for $\sum_{i=1}^{n} f(S_{-i}, O_i)$. We derive it as follows: consider an element $u \in E_{L,G}$ for some sets L, G. In S, the element u appears in $|L|$ sets. If $i \in L \cap G$ or $i \notin L \cup G$, it also appears in $|L|$ sets in (S_{-i}, O_i). Finally, if $i \in L \setminus G$, it appears in $|L| - 1$ sets in (S_{-i}, O_i). If $i \in G \setminus L$, it appears in $|L| + 1$ sets in (S_{-i}, O_i).

▶ **Theorem 5.** *If S is a δ-approximate local optimum, then $(1 + C_0\delta n \log n) w(\bigcup S) \geq (1 - 1/e^{[n]}) w(\bigcup O)$, for some universal constant C_0.*

Proof. Reformulating inequality (2) in terms of our symmetric notation and simplifying, we obtain

$$0 \leq \sum_{l,c,g} ((l + g + \delta n)\alpha_{l+c} - l\alpha_{l+c-1} - g\alpha_{l+c+1}) x_{l,c,g}. \tag{3}$$

Similarly, reformulating the statement of the theorem in terms of our symmetric notation, we obtain

$$0 \leq (1 + C_0\delta n \log n) \sum_{l+c\geq 1} x_{l,c,g} - \left(1 - \frac{1}{e^{[n]}}\right) \sum_{g+c\geq 1} x_{l,c,g}. \tag{4}$$

Since we have $x_{l,c,g} \geq 0$ for all l, c, g, in order to prove the lemma, it suffices to show that the coefficient of any term $x_{l,c,g}$ in (3) is at most its coefficient in (4). We consider three cases, and simplify expressions using the fact that $\alpha_0 = 0$. In the first, suppose that $g = c = 0$. We must show that for all $1 \leq l \leq n$,

$$(l + \delta n)\alpha_l - l\alpha_{l-1} \leq 1 + C_0\delta n \log n. \tag{5}$$

In the next case, suppose that $l = c = 0$. We must show that for all $1 \leq g \leq n$,

$$-g\alpha_1 \leq -\left(1 - \frac{1}{e^{[n]}}\right). \tag{6}$$

Finally, we must show for l, g, c such that $l + c \neq 0$, $g + c \neq 0$, and $1 \leq l + c + g \leq n$,

$$(l + g + \delta n)\alpha_{l+c} - l\alpha_{l+c-1} - g\alpha_{l+c+1} \leq \frac{1}{e^{[n]}} + \delta C_0 n \log n. \tag{7}$$

We now verify each of these inequalities, using the properties of α_i stated in Lemmas 2, 3, and 4. For (5), we have

$$\begin{aligned}(l + \delta n)\alpha_l - l\alpha_{l-1} &= l\beta_{l-1} + \delta n\alpha_l = \beta_l + \frac{1}{e^{[n]}} + \delta n\alpha_l \\ &\leq \beta_0 + \frac{1}{e^{[n]}} + \delta n\alpha_l = 1 + \delta n\alpha_l \leq 1 + C_0\delta n \log n.\end{aligned}$$

Inequality (6) follows directly from the fact that $g \geq 1$ and $\alpha_1 = 1 - \frac{1}{e^{[n]}}$. For inequality (7), we consider two cases. First, suppose $g = 0$ and so $c \geq 1$. Then, we have

$$(l+\delta n)\alpha_{l+c} - l\alpha_{l+c-1} \leq l\beta_l + \delta n\alpha_{l+c} = \beta_{l+1} + \frac{1}{e^{[n]}} - \beta_l + \delta n\alpha_{l+c}$$
$$\leq \frac{1}{e^{[n]}} + \delta n\alpha_{l+c} \;\leq\; \frac{1}{e^{[n]}} + C_0\delta n \log n.$$

If $g \geq 1$, then we have

$$(l+g+\delta n)\alpha_{l+c} - l\alpha_{l+c-1} - g\alpha_{l+c+1} = l\beta_{l+c-1} - g\beta_{l+c} + \delta n\alpha_{l+c}$$
$$\leq l\beta_{l+c-1} - \beta_{l+c} + \delta n\alpha_{l+c} = \frac{1}{e^{[n]}} - c\beta_{l+c} + \delta n\alpha_{l+c} \leq \frac{1}{e^{[n]}} + C_0\delta n \log n.$$

This completes the proof of Theorem 5. ◀

We obtain the following corollary by setting

$$\delta = \frac{\epsilon}{C_0 n \log n \left(1 - \frac{1}{e^{[n]}} - \epsilon\right)} = O\left(\frac{\epsilon}{n \log n}\right):$$

▶ **Corollary 6.** *Algorithm* LocalSearch$(C_1\epsilon/(n \log n))$ *is a* $(1 - 1/e^{[n]} - \epsilon)$*-approximation algorithm, for some universal constant* C_1 *and all* $\epsilon > 0$*.*

Since $e > e^{[n]}$, the same proof shows that if we replace $e^{[n]}$ by e in the definition of the sequence α_i, then the resulting approximation ratio is $1 - 1/e - \epsilon$.

Now, we turn to the run-time of Algorithm 2. First, we note that by keeping track of how many times each element of U is covered by the current solution, we can compute the change in f due to adding or removing a set from the solution using only $O(u)$ value oracle calls. The initial greedy phase takes time $O(nsu)$. Each iteration of the local search phase requires ns calls to the independence oracle for $\mathfrak{m}$ and $O(nsu)$ calls to the value oracle for w. Thus, each iteration can be completed in time $O(nsu)$. We now bound the total number of improvements that the algorithm can make.

▶ **Lemma 7.** *For any* $\delta > 0$*, Algorithm* LocalSearch(δ) *makes at most* $O(\delta^{-1})$ *improvements.*

Proof. Let $\hat{S}$ be the solution produced by the initial greedy phase of LocalSearch(δ), and let $\hat{O} = \arg\max_{S \in \mathfrak{m}} f(S)$. Then, LocalSearch$(\delta)$ makes at most $\log_{1+\delta}(f(\hat{O})/f(\hat{S}))$ improvements. Because the sequence of coefficients α_i is increasing and concave, for any $S \subseteq T \subset \mathcal{F}$ and $A \in \mathcal{F} \setminus T$ we must have $0 \leq f(T+A) - f(T) \leq f(S+A) - f(S)$, and so f is monotone submodular. Thus, the classical result of Fischer, Nemhauser, and Wolsey [10] implies that the greedy algorithm is a 2-approximation for maximizing f, so $2f(\hat{S}) \geq f(\hat{O})$. Algorithm LocalSearch$(\delta)$ can therefore make at most $\log_{1+\delta} 2 = \frac{\log 2}{\log(1+\delta)} = O(\delta^{-1})$ improvements. ◀

By setting $\epsilon = 1/e^{[n]} - 1/e$, we would obtain a clean $(1 - 1/e)$-approximation algorithm. However, since $1 - 1/e^{[n]}$ is very close to $1 - 1/e$, the resulting δ would be superpolynomial in n, and so we would not obtain a polynomial-time algorithm. Instead, we use a partial enumeration technique described by Khuller et al. [13]. Effectively, we try to "guess" a single set in the optimal solution, and then run LocalSearch on a contracted instance containing this set. We then iterate over all possible guesses.

▶ **Definition 8.** Let $\mathcal{I} = \langle U, w, \mathcal{F}, \mathfrak{m}\rangle$ be an instance of maximum coverage, where U is the universe, w is the weight function, $\mathcal{F}$ is a family of subsets of U, and $\mathfrak{m}$ is a matroid defined over $\mathcal{F}$.

Let $A \in \mathcal{F}$. The *contracted* instance $\mathcal{I}|_A = \langle U|_A, w|_A, \mathcal{F}|_A, \mathfrak{m}|_A \rangle$ is defined as follows:

$$U|_A = U,$$

$$w|_A = u \mapsto \begin{cases} w(u) & u \notin A, \\ 0 & u \in A, \end{cases}$$

$$\mathcal{F}|_A = \mathcal{F} - A,$$

$$\mathfrak{m}|_A = \mathfrak{m}/A = \{S \subset \mathcal{F} : S + A \in \mathfrak{m}\}.$$

Note that the definition of $w|_A$ directly implies that

$$w|_A\left(\bigcup S\right) = w\left(\bigcup S \setminus A\right) = w\left(\bigcup S \cup A\right) - w(A). \tag{8}$$

We can now formulate the new algorithm. Algorithm Approx simply runs LocalSearch(δ) on the instance $\mathcal{I}|_A$ for each $A \in \mathcal{F}$, and returns the best resulting solution of the original instance $\mathcal{I}$.

Algorithm 3 Approx(δ)

for $A \in \mathcal{F}$ **do**
  Let S_A be the result of running LocalSearch(δ) on instance $\mathcal{I}|_A$
end for
$A \leftarrow \arg\max_{A \in \mathcal{F}} w(\bigcup S_A \cup A)$
return $S_A + A$

▶ **Theorem 9.** *If* LocalSearch(δ) *has an approximation ratio of* θ *on matroids of rank* $n-1$ *then* Approx(δ) *has an approximation ratio of* $1/n + (1-1/n)\theta$ *on matroids of rank* n.

Proof. Let O be some optimal solution, and $A \in O$ be a set of maximum weight in O. Submodularity implies that $w(O) \leq \sum_{B \in O} w(B) \leq nw(A)$, and so $w(A) \geq w(O)/n$. Since $O - A \in \mathfrak{m}|_A$, we have $w|_A(\bigcup S_A) \geq \theta \cdot w|_A(\bigcup O \setminus A)$. From identity (8) we then have

$$\begin{aligned} w\left(\bigcup S_A \cup A\right) &= w(A) + w|_A\left(\bigcup S_A\right) \\ &\geq w(A) + \theta \cdot w|_A\left(\bigcup O \setminus A\right) \\ &= (1-\theta)w(A) + \theta \cdot w\left(\bigcup O\right) \\ &\geq \left(\frac{1-\theta}{n} + \theta\right) w\left(\bigcup O\right). \end{aligned}$$

◀

▶ **Corollary 10.** *For some universal constant* C_2 *and all* $n \geq 3$, *Algorithm* Approx($C_2/(n^2 \log n)$) *has an approximation ratio of at least* $1 - 1/e$.

Proof. Let $C_2 = 3C_1$, where C_1 is the constant defined in Corollary 6. The corollary implies that LocalSearch($C_2/(n^2 \log n)$) has an approximation ratio of $1 - 1/e^{[n-1]} - 1/3n$. Theorem 9 implies that the approximation ratio of Approx($C_2/(n^2 \log n)$) is

$$\frac{1}{n} + \left(1 - \frac{1}{n}\right)\left(1 - \frac{1}{e^{[n-1]}} - \frac{1}{3n}\right) \geq 1 - \frac{1}{e^{[n-1]}} + \frac{\frac{1}{e^{[n-1]}} - \frac{1}{3}}{n} \geq 1 - \frac{1}{e},$$

using the fact that $e^{[n-1]} \leq e^{[2]} = 3$. ◀

Our final algorithm Approx($C_2/(n^2 \log n)$) makes s calls to LocalSearch. Each of these calls makes at most $O(n^2 \log n)$ improvements, each taking time $O(nsu)$. The final runtime is thus $\tilde{O}(n^3 s^2 u)$.

References

1 Alexander A. Ageev and Maxim I. Sviridenko. Pipage rounding: a new method of constructing algorithms with proven performance guarantee. *J. of Comb. Opt.*, 8:17–30, 2004.

2 Paola Alimonti. New local search approximation techniques for maximum generalized satisfiability problems. In *Algorithms and Complexity*, volume 778 of *LNCS*, pages 40–53. Springer, 1994.

3 Richard A. Brualdi. Comments on bases in dependence structures. *Bull. Austral. Math. Soc.*, 1:161–167, 1969.

4 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. In *IPCO*, 2007.

5 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a monotone submodular function subject to a matroid constraint. In *STOC*, 2008.

6 Chandra Chekuri and Amit Kumar. Maximum coverage problem with group budget constraints and applications. In *APPROX*, pages 72–83, 2004.

7 Reuven Cohen and Liran Katzir. The generalized maximum coverage problem. *Inf. Proc. Lett.*, 108(1):15–22, 2008.

8 Gerard Cornuejols, Marshall L. Fisher, and George L. Nemhauser. Location of bank accounts to optimize float: An analytic study of exact and approximate algorithms. *Management Science*, 23(8):789–810, April 1977.

9 Uriel Feige. A threshold of $\ln n$ for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

10 M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey. An analysis of approximations for maximizing submodular set functions—II. In *Polyhedral Combinatorics*, volume 8 of *Mathematical Programming Studies*, pages 73–87. Springer Berlin Heidelberg, 1978.

11 Dorit S. Hochbaum and Anu Pathria. Analysis of the greedy approach in covering problems. *Naval Research Quarterly*, 45:615–627, 1998.

12 Sanjeev Khanna, Rajeev Motwani, Madhu Sudan, and Umesh V. Vazirani. On syntactic versus computational views of approximability. *SIAM J. Comp.*, 28:164–191, 1998.

13 Samir Khuller, Anna Moss, and Joseph (Seffi) Naor. The budgeted maximum coverage problem. *Inf. Process. Lett.*, 70:39–45, April 1999.

14 Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41:960–981, September 1994.

15 Andreas S. Schulz, James B. Orlin, and Abraham P. Punnen. Approximate local search in combinatorial optimization. *SIAM J. Comp.*, pages 587–596, 2004.

16 Aravind Srinivasan. Distributions on level-sets with applications to approximation algorithms. In *FOCS*, pages 588–599, 2001.

Trichotomy for Integer Linear Systems Based on Their Sign Patterns

Kei Kimura[1] **and Kazuhisa Makino**[2]

1 **Department of Mathematical Informatics, Graduate School of Information Science and Technology, University of Tokyo, Tokyo, 113-8656, Japan**
kei_kimura@mist.i.u-tokyo.ac.jp

2 **Department of Mathematical Informatics, Graduate School of Information Science and Technology, University of Tokyo, Tokyo, 113-8656, Japan**
makino@mist.i.u-tokyo.ac.jp

Abstract

In this paper, we consider solving the integer linear systems, i.e., given a matrix $A \in \mathbb{R}^{m\times n}$, a vector $b \in \mathbb{R}^m$, and a positive integer d, to compute an integer vector $x \in D^n$ such that $Ax \geq b$, where m and n denote positive integers, $\mathbb{R}$ denotes the set of reals, and $D = \{0, 1, \ldots, d-1\}$. The problem is one of the most fundamental NP-hard problems in computer science.

For the problem, we propose a complexity index η which is based only on the sign pattern of A. For a real γ, let $\mathrm{ILS}_=(\gamma)$ denote the family of the problem instances I with $\eta(I) = \gamma$. We then show the following trichotomy:

- $\mathrm{ILS}_=(\gamma)$ is linearly solvable, if $\gamma < 1$,
- $\mathrm{ILS}_=(\gamma)$ is weakly NP-hard and pseudo-polynomially solvable, if $\gamma = 1$, and
- $\mathrm{ILS}_=(\gamma)$ is strongly NP-hard, if $\gamma > 1$.

This, for example, includes the existing results that quadratic systems and Horn systems can be solved in pseudo-polynomial time.

1998 ACM Subject Classification G.2.1 Combinatorics

Keywords and phrases Integer linear system, Sign pattern, Complexity index, TVPI system, Horn system

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.613

1 Introduction

Integer linear systems

Let A denote a matrix $A \in \mathbb{R}^{m\times n}$, b denote a vector $b \in \mathbb{R}^m$, where m and n denote positive integers, and $\mathbb{R}$ denote the set of reals. For a positive integer d, let $D = \{0, 1, \ldots, d-1\}$. In this paper, we consider the problem of computing an integer vector $x \in D^n$ such that $Ax \geq b$, which we denote by ILS. The ILS problem is one of the most fundamental and important problems in computer science, and have been studied extensively from both theoretical and practical points of view [18, 26]. It is known that the ILS problem is strongly NP-hard, and can be solved in polynomial time, if m or n are bounded by some constant [22], or A is totally unimodular and b is integral [15]. When A is quadratic (also called TVPI, i.e., each row of A contains at most two nonzero elements) or Horn (i.e., each row of A contains at most one positive element), the ILS problem is known to be weakly NP-hard, but it can be solved in time polynomial in the input length and d, and hence in pseudo-polynomial time [20, 14, 29]. The best known bounds for quadratic and Horn systems require $O(md)$ time [2] and $O(n^2md)$ time, respectively. For unit linear systems, i.e., $A \in \{0, -1, +1\}^{m\times n}$, it is known that the

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 613–623
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

problem is still strongly NP-hard, but it can be solved in $O(nm)$ [21] and $O(n \log n + m)$ time [27] if A is in addition quadratic, and can be solved in $O(n^2 m)$ time [9, 28] if A is in addition Horn. Finally, for the difference constraint systems, i.e., $A \in \{0, -1, +1\}^{m \times n}$ and each row of A contains one positive element and one negative element, it is known that the problem is equivalent to the negative cycle detection in network theory and can be solved in $O(nm)$ [3, 11, 24] and $O(\sqrt{n}m \log C)$ [12], where C denotes the maximum absolute value of the negative elements in b.

A complexity index for integer linear systems

In this paper, we introduce a complexity index η for the ILS problem, which sharply distinguishes between the classes of *easy*, *semi-hard* and *hard* integer linear systems. The complexity index is based only on the sign pattern of A.

For a real a, its sign is defined as

$$\mathrm{sgn}(a) = \begin{cases} + & (a > 0) \\ 0 & (a = 0) \\ - & (a < 0), \end{cases} \tag{1}$$

and the sign of a real matrix $A \in \mathbb{R}^{m \times n}$ is the matrix $\mathrm{sgn}(A) \in \{0, -, +\}^{m \times n}$ which is obtained from A by replacing each element by its sign. For example, for a matrix

$$A = \begin{pmatrix} 1 & -3 & 0 \\ 4 & 2 & -5 \end{pmatrix}, \tag{2}$$

we have

$$\mathrm{sgn}(A) = \begin{pmatrix} + & - & 0 \\ + & + & - \end{pmatrix}. \tag{3}$$

Given an instance $I = (A, b, d)$ of the ILS problem, the index $\eta(I)$ is the optimal value of the following linear programming problem.

$$\begin{array}{lll} \text{min.} & Z & \\ \text{s.t.} & \sum_{j:\mathrm{sgn}(a_{ij})=+} \alpha_j + \sum_{j:\mathrm{sgn}(a_{ij})=-} (1 - \alpha_j) \le Z & (i = 1, \ldots, m) \\ & 0 \le \alpha_j \le 1 & (j = 1, \ldots, n). \end{array} \tag{4}$$

We note that neither numerical information of A, b nor d is used for our index $\eta(I)$, and it depends *only* on $\mathrm{sgn}(A)$, i.e., two problem instances I and I' have $\eta(I) = \eta(I')$ if the corresponding matrices have the same sign patterns.

The idea of this index originates from the works by Boros et al. [5], which introduced a complexity index for the Boolean satisfiability problem (SAT): Given a CNF $\varphi = \bigwedge_{i=1}^{m} \left(\bigvee_{j \in P_i} x_j \vee \bigvee_{j \in N_i} \overline{x}_j \right)$ of n variables, where $P_i, N_i \subseteq \{1, 2, \ldots, n\}$ with $P_i \cap N_i = \emptyset$, determine whether or not φ is satisfiable, i.e., whether or not there is $x \in \{0, 1\}^n$ such that $\varphi(x) = 1$. Their index distinguishes between the classes of easy and hard SAT instances. We can see that our index is a generalization of theirs to integer linear systems, since the Boolean satisfiability problem can be represented as integer linear systems with unit matrices $A \in \{0, -1, +1\}^{m \times n}$.

The results obtained in this paper

For a real γ, let $\mathrm{ILS}_{=}(\gamma)$ denote the family of the problem instances I with $\eta(I) = \gamma$. We then have the following main result.

▶ **Theorem 1.1.** (1) $\mathrm{ILS}_{=}(\gamma)$ *is linearly solvable, if* $\gamma < 1$,
(2) $\mathrm{ILS}_{=}(\gamma)$ *is weakly NP-hard and pseudo-polynomially solvable, if* $\gamma = 1$, *and*
(3) $\mathrm{ILS}_{=}(\gamma)$ *is strongly NP-hard, if* $\gamma > 1$.
Here we assume that $\mathrm{ILS}_{=}(\gamma) \neq \emptyset$ holds.

We also show that $\eta(I) < 1$, $= 1$, and > 1 can be checked in linear time. This theorem implies the existing results [2, 14, 29] that quadratic (i.e., TVPI) systems and Horn systems can be solved in pseudo-polynomial time, since quadratic systems and Horn systems are included in $\mathrm{ILS}_{=}(\gamma)$ with $\gamma \leq 1$, which will be discussed later.

If we restrict integer linear systems to Boolean satisfiability problem, then Boros et al. [5] showed that $\mathrm{ILS}_{=}(\gamma)$ is linearly solvable if $\gamma \leq 1$, and $\mathrm{ILS}_{=}(\gamma)$ is strongly NP-hard if $\gamma > 1$. Instead of their result, which partitions the SAT problem into *two* classes of *easy* and *hard* SAT instances, we partition integer linear systems into *three* classes of *easy*, *semi-hard* and *hard* systems.

For unit linear systems, i.e., $A \in \{0, -1, +1\}^{m \times n}$, we have the following result.

▶ **Theorem 1.2.** *Let* A *be a unit matrix, i.e.,* $A \in \{0, -1, +1\}^{m \times n}$. *Then we have*
(1) $\mathrm{ILS}_{=}(\gamma)$ *is polynomially solvable if* $\gamma \leq 1$.
(2) $\mathrm{ILS}_{=}(\gamma)$ *is strongly NP-hard if* $\gamma > 1$.

We note that Theorem 1.2 includes polynomial solvability for Horn and quadratic unit systems [1, 8, 9, 17, 21, 27, 28], and tractability of SAT problem (i.e., the satisfiability problem for 2-, Horn, renamable Horn, and q-Horn CNFs can be solved in polynomial time) [10, 13, 23, 4].

We generalize the results above by considering nonconstant γ. More precisely, we regard γ as a function of the number of variables n and d, and for such γ, let $\mathrm{ILS}_{\leq}(\gamma)$ denotes the family of the problem instances I with $\eta(I) \leq \gamma$. We have the following results.

▶ **Theorem 1.3.** (1) $\mathrm{ILS}_{\leq}(\gamma)$ *is linearly solvable, if* $\gamma < 1$.
(2) $\mathrm{ILS}_{\leq}(\gamma)$ *is weakly NP-hard and pseudo-polynomially solvable, if* $1 \leq \gamma \leq 1 + \frac{c \log_d n}{n}$ *for some constant* $c > 0$.
(3) $\mathrm{ILS}_{\leq}(\gamma)$ *is strongly NP-hard, if* $\gamma \geq 1 + \frac{1}{n^\delta}$ *for some constant* $\delta < 1$.

▶ **Theorem 1.4.** *Let* A *be a unit matrix, i.e.,* $A \in \{0, -1, +1\}^{m \times n}$. *Then we have*
(1) $\mathrm{ILS}_{\leq}(\gamma)$ *is polynomially solvable, if* $\gamma \leq 1 + \frac{c \log_d n}{n}$.
(2) $\mathrm{ILS}_{\leq}(\gamma)$ *is strongly NP-hard, if* $\gamma \geq 1 + \frac{1}{n^\delta}$ *for some constant* $\delta < 1$.

Finally, we mention that there exists a line of research for *sign solvability* for linear systems [7, 25], linear programming problem [16], and linear complementarity problem [19]. They mainly study sign patterns of the input data, that always determine sign patterns of solutions. Their works are motivated by the fact that the input data are uncertain but the structural properties are preserved in most practical situations. While both their and our works concern the sign patterns of the input, ours differs from theirs in that our work studies the *integer* solutions and does not concern sign patterns of the solutions.

2 Integer linear systems with index smaller than 1

For a given problem instance $I = (A, b, d)$, we denote by $(Z, \alpha_1, \ldots, \alpha_n)$ an optimal solution of (4). Let $V = \{1, \ldots, n\}$. In this paper, we assume without loss of generality that each variable is not redundant, i.e., A contains no column whose elements are all 0, since otherwise we can fix all redundant variables to 0, for example.

In this section, we consider the case in which $\eta(I) < 1$, i.e., $Z < 1$, and prove (1) in Theorem 1.3, which implies Theorems 1.1, 1.2, 1.4 when $\eta(I) < 1$.

2.1 The case of $\eta(I) < 1/2$

Let us first consider the case in which $Z = \eta(I) < 1/2$. Then, there exists no $j \in V$ with $\alpha_j = 1/2$, since otherwise we have $Z \geq 1/2$, a contradiction. If $\alpha_j > 1/2$ for some $j \in V$, then by $Z < 1/2$, the j-th column of A is nonpositive. Similarly, $\alpha_j < 1/2$ implies that the j-th column of A is nonnegative. These imply that $Z = 0$, $\alpha_j > 1/2 \Rightarrow \alpha_j = 1$, and $\alpha_j < 1/2 \Rightarrow \alpha_j = 0$. Therefore, we have the following lemma.

▶ **Lemma 2.1.** *If Problem* (4) *has the optimal value $Z < 1/2$, then we have $Z = 0$, and there exists a unique* 0-1 *optimal solution for* (4)*.*

Moreover, $\eta(I) < 1/2$ (and hence $\eta(I) = 0$) holds if and only if each column of A is either nonnegative or nonpositive. Let y be a n-dimensional vector such that $y_j = d - 1$ if j-th column of A is nonnegative, and 0, otherwise (i.e., if j-th column of A is nonpositive). Then it is not difficult to see that there exists a vector $x \in D^n$ with $Ax \geq b$ if and only if y satisfies $Ay \geq b$. These lead to the following lemma.

▶ **Lemma 2.2.** *Let $I = (A, b, d)$ be a problem instance. Then we can check whether $\eta(I) < 1/2$ in linear time, and if so, the problem can be solved in linear time.*

2.2 The case of $\eta(I) = 1/2$

We next consider the case in which $Z = \eta(I) = 1/2$.

If $\alpha_j > 1/2$ (resp., $\alpha_j < 1/2$) for some $j \in V$, then $Z = 1/2$ implies that the j-th column of A is nonpositive (resp., nonnegative). Define a vector $\alpha^* \in \mathbb{R}^n$ by $\alpha^*_j = 1$ if the j-th column of A is nonpositive, 0 if the j-th column of A is nonnegative, and $1/2$, otherwise. Then we can see that this α^* is also an optimal solution of (4).

▶ **Lemma 2.3.** *If Problem* (4) *has the optimal value $Z = 1/2$, then it has a half-integral optimal solution.*

Moreover, $\alpha^*_j = 1/2$ if and only if the j-th column of A contains both positive and negative elements, and if $a_{ij} \neq 0$ for such j, then the i-th row of A contains no nonzero element a_{ik} with $k \neq j$ and $\alpha^*_k = 1/2$. Let us fix $x_j = 0$ for all $j \in V$ with $\alpha^*_j = 1$, and $x_j = d - 1$ for all $j \in V$ with $\alpha^*_j = 0$. Then each inequality of the resulting integer linear system contains at most one variable, and hence it can be easily solved.

▶ **Lemma 2.4.** *Let $I = (A, b, d)$ be a problem instance. Then we can check whether $\eta(I) = 1/2$ in linear time, and if so, the problem can be solved in linear time.*

2.3 The case of $1/2 < \eta(I) < 1$

In this section, we consider the case in which $1/2 < Z = \eta(I) < 1$. Note that in this case Problem (4) might have no (half-)integral optimal solution. For example, let A be a $(n+1) \times n$ matrix such that $a_{ij} = -1$ if $i = j$, 1 if $i = n + 1$, and 0 otherwise. Then the problem has a unique optimal solution $Z = \frac{n}{n+1}$ and $\alpha_j = \frac{1}{n+1}$ for all j.

For a subset $S \subset \mathbb{R}$, let $V_S = \{j \in V \mid \alpha_j \in S\}$. For two reals a and b with $a < b$, $[a, b) = \{z \in \mathbb{R} \mid a \leq z < b\}$, $(a, b] = \{z \in \mathbb{R} \mid a < z \leq b\}$ and $[a, b] = \{z \in \mathbb{R} \mid a \leq z \leq b\}$. Let ε be a positive number that satisfies $Z \leq 1 - \varepsilon$ and $2k\varepsilon = 1$ for some integer k, where we note that ε might depend on m and n. We then partition $[0, 1]$ into $2k + 1$ sets

$$[0,1] = \bigcup_{\ell=1}^{k} [(\ell-1)\varepsilon, \ell\varepsilon) \cup \{1/2\} \cup \bigcup_{\ell=1}^{k} (1-\ell\varepsilon, 1-(\ell-1)\varepsilon]. \tag{5}$$

For $i = 1, 2, \ldots, m$, let $P_i = \{j \in V \mid a_{ij} > 0\}$ and $N_i = \{j \in V \mid a_{ij} < 0\}$. Then we have the following properties.

▶ **Lemma 2.5.** *Let $I = (A, b, d)$ be a problem instance with $1/2 < \eta(I) < 1$, and let ε be defined as above. Then*

(i) *$V_{(1-\varepsilon,1]} \cap P_i = \emptyset$ and $V_{[0,\varepsilon)} \cap N_i = \emptyset$ hold for all $i = 1, 2, \ldots, m$.*

(ii) *If $j \in V_{(1-(\ell+1)\varepsilon,1-\ell\varepsilon]} \cap P_i$ for some $\ell = 1, 2, \ldots, k$ and $i = 1, 2, \ldots, m$, then we have $P_i - \{j\} \subseteq V_{[0,\ell\varepsilon)}$ and $N_i \subseteq V_{(1-\ell\varepsilon,1]}$.*

(iii) *If $j \in V_{[\ell\varepsilon,(\ell+1)\varepsilon)} \cap N_i$ for some $\ell = 1, 2, \ldots, k$ and $i = 1, 2, \ldots, m$, then we have $P_i \subseteq V_{[0,\ell\varepsilon)}$ and $N_i - \{j\} \subseteq V_{(1-\ell\varepsilon,1]}$.*

Proof. (i), (ii), and (iii) follow from $Z \leq 1 - \varepsilon$. ◀

By (i) in Lemma 2.5, if $j \in V_{(1-\varepsilon,1]}$, then the j-th column of A is nonpositive, and hence we can fix $x_j = 0$. Similarly, if $j \in V_{[0,\varepsilon)}$, then the j-th column of A is nonnegative, and hence we can fix $x_j = d - 1$. After fixing variables in $V_{(1-\varepsilon,1]} \cup V_{[0,\varepsilon)}$, if $a_{ij} > 0$ for some $j \in V_{(1-2\varepsilon,1-\varepsilon]}$, then (ii) in Lemma 2.5 implies that the i-th inequality of the resulting system contains only one variable x_j. By solving such inequalities, we have a lower bound $x_j \geq p_j\,(\in D)$. Since all the other inequalities have $a_{ij} \leq 0$, we can fix $x_j = p_j$. Similarly, if $a_{ij} < 0$ for some $j \in V_{[\varepsilon,2\varepsilon)}$, then (iii) in Lemma 2.5 implies that the i-th inequality of the resulting system contains only one variable x_j. By solving such inequalities, we have an upper bound $x_j \leq p_j\,(\in D)$. Since all the other inequalities have $a_{ij} \geq 0$, we can fix $x_j = p_j$. By repeatedly applying this argument to variables in $V_{(1-(\ell+1)\varepsilon,1-\ell\varepsilon]}$ and $V_{[\ell\varepsilon,(\ell+1)\varepsilon)}$ for $\ell = 2, 3, \ldots k$, we can fix all the variables in $V \setminus V_{\{1/2\}}$. Note that by (ii) and (iii) in Lemma 2.5, each inequality of the resulting system consists of at most one variable. Hence we can solve it in linear time.

Formally, we describe the algorithm in Algorithm 2.7. We note that the algorithm uses no information of $(Z, \alpha_1, \ldots, \alpha_n)$ of (4).

We remark that if the algorithm above solved the integer linear system, then we have $\eta(I) < 1$. Since we can check whether $\eta(I) \leq 1/2$ in linear time by Lemmas 2.2 and 2.4, we have the following result.

▶ **Lemma 2.6.** *Let $I = (A, b, d)$ be a problem instance. Then we can check whether $1/2 < \eta(I) < 1$ in linear time, and if so, the problem can be solved in linear time.*

By combining Lemmas 2.2, 2.4, and 2.6, we have (1) in Theorem 1.3.

▶ **Algorithm 2.7.**

Step 1.
for $1 \leq j \leq n$ **do**
if *j-th column of A is nonpositive* **then** $x_j := 0$
else if *j-th column of A is nonnegative* **then** $x_j := d - 1$
end if
end for
if *the resulting system has an inconsistent inequality (with no variable)* **then** *output "infeasible" and halt*
else *remove inequalities with no variable from the system*
end if

Step 2.
while *the resulting system has $j \in V$ such that $a_{ij'} = 0$ for all i and j' with $a_{ij} > 0$ and $j' \neq j$* **do**
compute a lower bound $x_j \geq p$ by solving inequalities in $\{i \mid a_{ij} > 0\}$
if $p \leq d$ **then** $x_j := \max\{\lceil p \rceil, 0\}$
else *output "infeasible" and halt*
end if
if *the resulting system has an inconsistent inequality (with no variable)* **then** *output "infeasible" and halt*
else *remove inequalities with no variable from the system*
end if
end while
Step 3.
while *the resulting system has $j \in V$ such that $a_{ij'} = 0$ for all i and j' with $a_{ij} < 0$ and $j' \neq j$* **do**
compute an upper bound $x_j \leq p$ by solving inequalities in $\{i \mid a_{ij} < 0\}$
if $p \geq 0$ **then** $x_j := \min\{\lfloor p \rfloor, d - 1\}$
else *output "infeasible" and halt*
end if
if *the resulting system has an inconsistent inequality (with no variable)* **then** *output "infeasible" and halt*
else *remove inequalities with no variable from the system*
end if
end while
Step 4. */* Note that each inequalities of the resulting system has exactly one variable.*/*
Solve the resulting system.

It is not difficult to see that the algorithm 2.7 above can be implemented in linear time in the input length and the number of nonzero elements of A.

3 Integer linear systems with index 1

In this section, we assume that integer linear systems have index 1, and prove Theorems 1.1 and 1.2 for this case.

Let $(Z, \alpha_1, \ldots, \alpha_n)$ be an optimal solution of (4). Then we note that $|P_i \cap V_{(1/2,1]}| \leq 1$, $|N_i \cap V_{[0,1/2)}| \leq 1$, and $|(P_i \cup N_i) \cap V_{\{1/2\}}| \leq 2$ holds for all $i = 1, 2, \ldots m$, since

otherwise we have $Z > 1$, a contradiction. Moreover, $(P_i \cup N_i) \cap V_{\{1/2\}} \neq \emptyset$ implies $P_i \cap V_{(1/2,1]}, N_i \cap V_{[0,1/2)} = \emptyset$, which again follows from $Z = 1$. Define a vector $\alpha^* \in \mathbb{R}^n$ by $\alpha^*_j = 0$ if $\alpha_j < 1/2$, $\alpha^*_j = 1/2$ if $\alpha_j = 1/2$, and $\alpha^*_j = 1$, otherwise (i.e., if $\alpha_j > 1/2$). It is not difficult to see that α^* is also an optimal solution of (4).

▶ **Lemma 3.1** ([5]). *If Problem* (4) *has the optimal value* $Z = 1$*, then it has a half-integral optimal solution.*

Moreover, such a solution can be computed in linear time.

▶ **Lemma 3.2** ([6]). *We can decide whether Problem* (4) *has the optimal value* $Z = 1$ *in linear time, and if so, we can compute a half-integral optimal solution in linear time.*

Let $\alpha \in \{0, 1/2, 1\}^n$ denote an optimal solution of Problem (4). To make discussion clear, we may assume $\alpha \in \{1/2, 1\}^n$ without loss of generality. To see this, assume that $\alpha_j = 0$ holds for some j. We then replace the variable x_j to a new variable $x'_j\,(= d - 1 - x_j)$, i.e., we substitute $x_j := d - 1 - x'_j$ in the system. It is not difficult to see that the feasibility of the original integer linear system is equivalent to the one of the resulting system. Since the coefficient matrix of the resulting system differs A only by the sign of the j-th column of matrix A, we have a half-integral optimal solution with $\alpha_j = 1$ for the new LP problem (4). By replacing all variables j with $\alpha_j = 0$, we have the integer linear system such that problem (4) has an optimal solution $\alpha \in \{1/2, 1\}^n$. We remark that this replacement can be done in linear time.

Let $Q = V_{\{1/2\}}$ and $H = V_{\{1\}}$. By $\alpha \in \{1/2, 1\}^n$, V can be partitioned into Q and H:

$$V = Q \cup H. \tag{6}$$

Then by the discussion at the beginning of this section, we have the following properties.

▶ **Lemma 3.3** (QH-partition [5]). *A partition* $V = Q \cup H$ *satisfies the following three conditions:*

(a) *Each row* i *of* A *contains at most two nonzero elements* a_{ij} *with* $j \in Q$*. Or equivalently,* $|(P_i \cup N_i) \cap Q| \leq 2$ *holds for all* $i = 1, 2, \ldots, m$*.*

(b) *Each row* i *of* A *contains at most one positive element* a_{ij} *with* $j \in H$*. Or equivalently,* $|P_i \cap H| \leq 1$ *holds for all* $i = 1, 2, \ldots, m$*.*

(c) *If row* i *of* A *contains a positive element* a_{ij} *with* $j \in H$*, then the elements* a_{ik} *with* $k \in Q$ *are all zeros. Or equivalently,* $P_i \cap H \neq \emptyset \Rightarrow (P_i \cup N_i) \cap Q = \emptyset$ *for all* $i = 1, 2, \ldots, m$*.*

For a QH-partition of V, let S denote the set of rows i of A such that $a_{ij} = 0$ for all $j \in Q$. Let $A[S, H]$ denote the submatrix of A whose row and column sets are S and H, respectively, and let b_H and x_H respectively denote the restriction of b and x to H. Then by Lemma 3.3 (a), linear system $A[S, H]x_H \geq b_H$ is Horn, i.e., each row of $A[S, H]$ contains at most one positive element. It is known that any Horn system has a unique minimal solution if it is feasible. Let $x^*_H \in D^H$ be such a solution for $A[S, H]x_H \geq b_H$. Since Lemma 3.3 (c) implies that any element a_{ij} with $i \notin S$ and $j \in H$ is nonpositive, we can see that the original integer linear system is feasible if and only if so is the system obtained from it by substituting $x_H = x^*_H$. Thus we consider the system obtained by fixing $x_H = x^*_H$. Since the resulting system is quadratic (i.e., each row contains at most two nonzero elements), we can solve it, for example, by the algorithm proposed in [14]. We summarize this algorithm in Algorithm 3.4.

▶ **Algorithm 3.4.**

Step 1.
Compute a QH-partition of V

Step 2.
if *the integer linear system $x_H \in D^H$ and $A[S,H]x_H \geq b_H$ is infeasible* **then** *output "infeasible" and halt*
else *compute a unique minimal solution $x^*_H \in D^H$ of the system and substitue $x_H := x^*_H$*
end if

Step 3.
if *the resulting system is infeasible* **then** *output "infeasible" and halt*
else *compute an integer solution $x^*_Q \in D^Q$ of the resulting system, and output the vector (x^*_H, x^*_Q) and halt*
end if

▶ **Lemma 3.5.** *Algorithm 3.4 solves the integer linear system with index* 1 *in time polynomial in n, m and d.*

Proof. Since the correctness of algorithm 3.4 follows from the discussion before the description of the algorithm, we discuss its time complexity only.

By [6], Step 1 can be executed in linear time. Steps 2 and 3 can be done in polynomial time in n, m, and d [29, 14]. Therefore, in total, the algorithm requires polynomial time in n, m, and d. ◀

▶ **Lemma 3.6.** *For unit matrix A, Algorithm 3.4 solves the integer linear system with index* 1 *in polynomial time.*

Proof. The lemma follows from the fact that Horn and quadratic integer linear systems are solvable in polynomial time, if A is unit [8, 17]. ◀

We next show the weak NP-hardness of the problem.

▶ **Lemma 3.7.** $\mathrm{ILS}_=(1)$ *is weakly NP-hard.*

Proof. It is known [20] that solving Horn or quadratic system is weakly NP-hard. We show that Horn and quadratic systems both have index at most 1. Since the integer linear system with index less than 1 is solvable in linear time, this proves the lemma.

Let $I = (A, b, d)$ be a Horn system. Then we assign all the variables α_j to 1. Since each row of A contains at most one positive element, we have $\eta(I) \leq 1$. On the other hand if I is quadratic, then by assigning all the variables α_j to $1/2$, we have $\eta(I) \leq 1$, since each row of A contains at most two nonzero elements. ◀

4 Integer linear systems with index η with $1 < \eta \leq 1 + \frac{c \log_d n}{n}$

In this section, we consider the case in which $1 < \eta(I) \leq 1 + \frac{c \log_d n}{n}$, and complete the proof of (2) in Theorem 1.3 and (1) in Theorem 1.4. Our positive results can be regarded as generalizations of the ones for $\mathrm{ILS}_=(1)$.

A partition of V into Q, H, and Y, i.e., $V = Q \cup H \cup Y$ is called QHY-partition, if Q and H satisfy all the conditions in Lemma 3.3.

If we have a QHY-partition with small Y, then the integer linear system can be solved by assigning all possible assignments to variables in Y. For this purpose, we make use of the following result.

▶ **Lemma 4.1** ([5]). *A QHY-partition with $|Y| < 6n(\eta(I)-1)$ can be computed in polynomial time.*

By using this lemma, if γ is a function of n with $\gamma \leq 1 + \frac{c \log_d n}{n}$, then we have a QHY-partition with $|Y| \leq 6c \log_d n$. Note that each of the $d^{|Y|}$ assignments to the variables of Y produces a problem instance I^* with $\eta(I^*) \leq 1$. Each such instance is solvable in pseudo-polynomial time by Lemma 3.5, and if A is unit, it is solved in polynomial time by Lemma 3.6. Moreover, since $d^{|Y|} \leq n^{6c}$, we have that the integer linear systems can be solved in pseudo-polynomial time if the system has index at most $1 + \frac{c \log_d n}{n}$ for some constant c, and in polynomial time if the system is in addition unit.

5 Strong NP-hardness for integer linear systems

In this section, we show the strong NP-hardness for the integer linear systems, i.e., we prove (2) in Theorems 1.2 and 1.4, which implies (3) in Theorems 1.1 and 1.3.

We first show that $\mathrm{ILS}_{\leq}(\gamma)$ is NP-hard, if $\gamma \geq 1 + \frac{1}{n^\delta}$ for some constant $\delta < 1$. To do this, we reduce the Boolean satisfiability problem (SAT) to our problem.

Given a CNF $\varphi = \bigwedge_{i=1}^{m} \left(\bigvee_{j \in P_i} x_j \vee \bigvee_{j \in N_i} \overline{x}_j \right)$, we construct an integer linear system as follows:

$$\begin{array}{rcll} \sum_{j \in P_i} x_j + \sum_{j \in N_i} (1 - x_j) & \geq & 1 & (i = 1, \ldots, m) \\ x & \in & \{0,1\}^n. & \end{array} \tag{7}$$

Namely, A is a matrix defined by

$$a_{ij} = \begin{cases} 1 & j \in P_i \\ -1 & j \in N_i \\ 0 & \text{otherwise,} \end{cases} \tag{8}$$

b is a vector defined by

$$b_i = 1 - |N_i| \qquad (i = 1, \ldots, m), \tag{9}$$

and $d = 2$.

It is not difficult to see that φ is satisfiable if and only if there exists a $x \in D^n$ such that $Ax \geq b$. Since this reduction is polynomial, solving the integer linear system is in general NP-hard. Moreover, as mentioned in the introduction, our index η is a generalization of the complexity index of SAT defined by Boros et al. [5].

▶ **Lemma 5.1.** *Let $Z(\varphi)$ denote the complexity index of CNF φ defined in* [5]*, and $\eta(I)$ denote the complexity index of the integer linear system defined as* (7)*. Then we have $Z(\varphi) = \eta(I)$.*

We now refer the following theorem due to Boros et al. [5], where $\mathrm{SAT}(\gamma)$ denotes the set of instances φ of SAT such that $Z(\varphi) \leq \gamma$.

▶ **Theorem 5.2** ([5]). *SAT(γ) is strongly NP-hard, if $\gamma \geq 1 + \frac{1}{n^\delta}$ for some constant $\delta < 1$.*

By combining Theorem 5.2 with Lemma 5.1, we have the following result.

▶ **Lemma 5.3.** *Let γ be a function of n such that $\gamma \geq 1 + \frac{1}{n^\delta}$ for some constant $\delta < 1$. Then $\mathrm{ILS}_{\leq}(\gamma)$ is strongly NP-hard, even if A is unit.*

Note that Lemma 5.3 implies that for any constant $\gamma > 1$, $\mathrm{ILS}_{\leq}(\gamma)$ is NP-hard, even if A is unit. In order to show (2) in Theorems 1.2, we consider the following simple reduction.

Let A (resp., A') be a unit $m \times n$ (resp., $m' \times n'$) matrix with the optimal value γ (resp., γ') of (4). Consider the following integer linear system:

$$\begin{pmatrix} A & 0 \\ 0 & A' \end{pmatrix} \begin{pmatrix} x \\ x' \end{pmatrix} \geq \begin{pmatrix} 0 \\ b' \end{pmatrix},$$

where 0 denote a zero matrix (or vector) of appropriate size, and b' denote a vector in $\mathbb{R}^{m'}$. We can see that this system has a solution if and only if $A'x' \geq b'$ has a solution, since $x = 0$ clearly satisfies $Ax \geq 0$. If we choose $A'x' \geq b'$ from strongly NP-hard instances with $\gamma' \leq \gamma$, we have the following results.

▶ **Lemma 5.4.** *Let γ be a constant with $\gamma > 1$ and $\mathrm{ILS}_{=}(\gamma) \neq \emptyset$. Then $\mathrm{ILS}_{=}(\gamma)$ is strongly NP-hard, even if A is unit.*

Acknowledgment

The authors are grateful to the reviewers for their comments and suggestions.

References

1 Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. An improved tight closure algorithm for integer octagonal constraints. In *Proceedings of the 9th International Conference on Verification, Model Checking and Abstract Interpretation*, pages 8–21, 2008.

2 Reuven Bar-Yehuda and Dror Rawitz. Efficient algorithms for integer programs with two variables per constraint. *Algorithmica*, 29(4):595–609, 2001.

3 Richard Ernest Bellman. On a routing problem. *Quarterly Applied Mathematics*, 16:87–90, 1958.

4 Endre Boros, Yves Crama, and Peter L. Hammer. Polynomial-time inference of all valid implications for Horn and related formulae. *Annals of Mathematics and Artificial Intelligence*, 1:21–32, 1990.

5 Endre Boros, Yves Crama, Peter L. Hammer, and Michael E. Saks. A complexity index for satisfiability problems. *SIAM Journal on Computing*, 23(1):45–49, 1994.

6 Endre Boros, Peter L. Hammer, and Xiaorong Sun. Recognition of q-Horn formulae in linear time. *Discrete Applied Mathematics*, 55:1–13, 1994.

7 Richard A. Brualdi and Bryan L. Shader. *Matrices of Sign-Solvable Linear Systems*. Cambridge University Press, Cambridge, 1995.

8 R. Chandrasekaran. Integer programming problems for which a simple rounding type algorithm works. *Progress in Combinatorial Optimization*, 8:101–106, 1984.

9 R. Chandrasekaran and K. Subramani. A combinatorial algorithm for horn programs. In *Proceedings of the 20th International Symposium on Algorithms and Computation*, pages 1114–1123, 2009.

10 Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM Journal on Computing*, 5(4):691–703, 1976.

11 Lester Randolph Ford. Network flow theory. Technical report, The Rand Corporation, Santa Monica, 1956.

12 Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24:494–504, 1995.

13 Lawrence J. Henschen and Lawrence A. Wos. Unit refutations and horn sets. *Journal of ACM*, 21:590–605, 1974.

14 Dorit S. Hochbaum, Nimrod Megiddo, Joseph Naor, and Arie Tamir. Tight bounds and 2-approximation algorithms for integer programs with two variables per inequality. *Mathematical Programming*, 62:69–83, 1993.

15 Alan J. Hoffman and Joseph B. Kruskal. Integral boundary points of convex polyhedra. In H.W. Khun and A.J. Tucker, editors, *Linear Inequalities and Related Systems*, pages 223–246. Princeton University Press, 1956.

16 Satoru Iwata and Naonori Kakimura. Solving linear programs from sign patterns. *Mathematical Programming*, 114:393–418, 2008.

17 Joxan Jaffar, Michael J. Maher, Peter J. Stuckey, and Roland H. C. Yap. Beyond finite domains. *Principles and Practice of Constraint Programming*, 874:86–94, 1994.

18 Michael Jünger, Thomas M. Liebling, Denis Naddef, George L. Nemhauser, William R. Pulleyblank, Gerhard Reinelt, Giovanni Rinaldi, and Laurence A. Wolsey, editors. *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Springer-Verlag, Heidelberg, 2010.

19 Naonori Kakimura. Sign-solvable linear complementarity problems. *Linear Algebra and Its Applications*, 429:606–616, 2008.

20 J. C. Lagarias. The computational complexity of simultaneous Diophantine approximation problems. *SIAM Journal on Computing*, 14(1):196–209, 1985.

21 Shuvendu K. Lahiri and Madanlal Musuvathi. An efficient decision procedure for UTVPI constraints. In *Proceedings of the 5th International Workshop on Frontiers of Combining Systems*, pages 168–183, 2005.

22 Hendrik Willem Lenstra, Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8:538–548, 1983.

23 Harry R. Lewis. Renaming a set of clauses as a horn set. *Journal of the ACM*, 25:134–135, 1978.

24 Edward Forrest Moore. The shortest path through a maze. In *the International Symposium on the Theory of Switching*, 1959.

25 Paul A. Samuelson. *Foundations of Economic Analysis*. Harvard University Press, 1947.

26 Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons Ltd, Chichester, 1986.

27 Andreas Schutt and Peter J. Stuckey. Incremental satisfiability and implication for UTVPI constraints. *INFORMS Journal of Computing*, 22:514–527, 2010.

28 K. Subramani and James Worthington. A new algorithm for linear and integer feasibility in horn constraints. In *Proceedings of the 8th International Conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming*, pages 215–229, 2011.

29 Hans van Maaren and Chuangyin Dang. Simplicial pivoting algorithms for a tractable class of integer programs. *Journal of Combinatorial Optimization*, 6(2):133–142, 2002.

Tying up the loose ends in fully LZW-compressed pattern matching*

Paweł Gawrychowski[1]

1 **Institute of Computer Science, University of Wrocław, Poland**
Max-Planck-Institute für Informatik, Saarbrücken, Germany
gawry@cs.uni.wroc.pl

Abstract

We consider a natural generalization of the classical pattern matching problem: given compressed representations of a pattern $p[1..M]$ and a text $t[1..N]$ of sizes m and n, respectively, does p occur in t? We develop an optimal linear time solution for the case when p and t are compressed using the LZW method. This improves the previously known $\mathcal{O}((n+m)\log(n+m))$ time solution of Gąsieniec and Rytter [11], and essentially closes the line of research devoted to studying LZW-compressed exact pattern matching.

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases pattern matching, compression, Lempel-Ziv-Welch

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.624

1 Introduction

One of the most natural problems concerning processing information is *pattern matching*, in which we are given a pattern $p[1..M]$ and a text $t[1..N]$, and have to check if there is an occurrence of p in t. Although many very efficient (both from a purely theoretical and more practically oriented point of view) solutions to this problem are known [16, 8, 13, 9, 5, 4], most data is archived and stored in a compressed form. This suggest an intriguing research direction: if the text, or both the pattern and the text, are given in their compressed representations, do we really need to decompress them in order to detect an occurrence? If just the text is compressed, this is the *compressed pattern matching* problem. For Lempel-Ziv-Welch compression, used for example in Unix `compress` utility and GIF images, Amir *et al.* introduced two algorithms with respective time complexities $\mathcal{O}(n+M^2)$ and $\mathcal{O}(n\log M + M)$ [1], where n is the compressed size of the text. The pattern preprocessing time was then improved [14] to get $\mathcal{O}(n+M^{1+\epsilon})$ time complexity. In a recent paper [12] we proved that in fact a $\mathcal{O}(n+M)$ solution is possible, as long as the alphabet consists of integers which can be sorted in linear time. A more general problem is the *fully compressed pattern matching*, where both the text and the pattern are compressed. This problem seems to be substantially more involved than compressed pattern matching, as we cannot afford to perform any preprocessing for every possible prefix/suffix of the pattern, and such preprocessing is a vital ingredient of any efficient pattern matching algorithm known to the author. Nevertheless, Gąsieniec and Rytter [11] developed a $\mathcal{O}((n+m)\log(n+m))$ time algorithm for this problem, where n and m are the compressed sizes of the text and the pattern, respectively.

* Supported by MNiSW grant number N N206 492638, 2010–2012 and START scholarship from FNP.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 624–635

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

In this paper we show that in fact an optimal linear time solution is possible for fully LZW-compressed pattern matching. The starting point of our algorithm is the $\mathcal{O}(n+M)$ time algorithm [12]. Of course we cannot afford to use it directly as M might be of order m^2. Nevertheless, we can apply the method to a few carefully chosen fragments of the pattern. Then, using those fragments we try to prune the set of possible occurrences, and verify them all at once by a combined usage of the so-called PREF function and iterative compression of the (compressed) pattern similar to the method of [11]. The chosen fragments correspond to the most complicated part of the pattern, in a certain sense. If there is no such part, we observe that the pattern is periodic, and a modification of the algorithm from [12] can be applied. To state this modification, and prove its properties, we briefly review the algorithm from [12] in the next section. While the modification itself might seem simple, we would like to point out that it is nontrivial, and we need quite a few additional ideas in order to get the result.

2 Preliminaries

We consider strings over finite alphabet Σ (which consists of integers which can be sorted in linear time, namely $\Sigma = \{1, 2, \ldots, (n+m)^c\}$) given in a Lempel-Ziv-Welch compressed form, where a string is represented as a sequence of *codewords*. Each codeword is either a single letter or a previously occurring codeword concatenated with a single character. This additional character is not given explicitly: we define it as the first character of the next codeword, and initialize the set of codewords to contain all single characters in the very beginning. The resulting compression method enjoys a particularly simple encoding/decoding process, but unfortunately requires outputting at least $\Omega(\sqrt{N})$ codewords (so, for example, we are not able to achieve an exponential compression possible in the general Lempel-Ziv method). Still, its simplicity and good compression ratio achieved on real life instances make it an interesting model to work with. For the rest of the paper we will use LZW when referring to Lempel-Ziv-Welch compression.

We are interested in a variation of the classical pattern matching problem: given a pattern $p[1\,..\,M]$ and a text $t[1\,..\,N]$, does p occur in t? We assume that both p and t are given in LZW compressed forms of size m and n, respectively, and wish to achieve a running time depending on the size of the compressed representation $n+m$, not on the original lengths N and M. If the pattern does occur in the text, we would like to get the position of its first occurrence. We call such problem the *fully compressed pattern matching*.

A closely related problem is the *compressed pattern matching*, where we aim to detect the first occurrence of an uncompressed pattern in a compressed text. In a previous paper [12], we proved that this problem can be solved in deterministic linear time. This $\mathcal{O}(n+M)$ time algorithm will be our starting point. Of course we cannot directly apply it as M might be of order m^2. Nevertheless, a modified version of this solution will be one of our basic building bricks. In order to state the modification and prove its correctness we briefly review the idea behind the original algorithm in the remaining part of this section. First we need a few auxiliary lemmas. A *period* of a word w is an integer $0 < d \leq |w|$ such that $w[i] = w[i+d]$ whenever both letters are defined.

▶ **Lemma 1** (Periodicity lemma). *If both d and d' are periods of w, and $d+d' \leq |w|+\gcd(d,d')$, then $\gcd(d,d')$ is a period as well.*

Using any linear time suffix array construction algorithm and fast LCA queries [2] we get the following.

▶ **Lemma 2.** *Pattern p can be preprocessed in linear time so that given any two fragments $p[i\,..\,i+k]$ and $p[j\,..\,j+k]$ we can find their longest common prefix (suffix) in constant time.*

▶ **Lemma 3.** *Pattern p can be preprocessed in linear time so that given any fragment $p[i\,..\,j]$ we can find its longest prefix which is a suffix of the whole pattern in constant time, assuming we know the (explicit or implicit) vertex corresponding to $p[i\,..\,j]$ in the suffix tree.*

A *border* of a word w is an integer $0 \leq b < |w|$ such that $w[1\,..\,b] = w[|w| - b + 1\,..\,|w|]$. The longest border is usually called the border (similarly, the shortest period is called the period). By applying the preprocessing from the Knuth-Morris-Pratt algorithm for both the pattern and the reversed pattern we get the following.

▶ **Lemma 4.** *Pattern p can be preprocessed in linear time so that we can find the border of each its prefix (suffix) in constant time.*

A *snippet* is any substring (factor) $p[i\,..\,j]$ of the pattern, represented as a pair of integers (i, j). If $i = 1$ we call it a *prefix snippet*, and if $j = |p|$ a *suffix snippet*. An *extended snippet* is a snippet for which we also store the corresponding vertex in the suffix tree (built for the pattern) and the longest suffix which is a prefix of the pattern. A sequence of snippets is a concatenation of a number of substrings of the pattern.

A high-level idea behind the linear time LZW-compressed pattern matching is to first reduce the problem to pattern matching in a sequence of extended snippets. It turns out that if the alphabet is of constant size, the reduction can be almost trivially performed in linear time, and for polynomial size integer alphabets we can apply more sophisticated tools to get the same complexity. Then we focus on pattern matching in a sequence of snippets. The idea is to simulate the well-known Knuth-Morris-Pratt algorithm while operating on whole snippets instead of single characters using Lemma 5 and Lemma 6.

▶ **Lemma 5.** *Given a prefix snippet and a suffix snippet we can detect an occurrence of the pattern in their concatenation in constant time.*

▶ **Lemma 6.** *Given a prefix snippet $p[1\,..\,i]$ and a snippet $p[j\,..\,k]$ we can find the longest long border b of $p[1\,..\,i]$ such that $p[1\,..\,b]p[j\,..\,k]$ is a prefix of the whole p in constant time, where a long border is $b \geq \frac{i}{2}$ such that $p[1\,..\,b] = p[i - b + 1\,..\,i]$.*

During the simulation we might create some new snippets, but they will be always either prefix snippets or *half snippets* of the form $p[\frac{i}{2}\,..\,i]$. All information required to make those snippets extended can be precomputed in a relatively straightforward way using $\mathcal{O}(M)$ time.

The running time of the resulting procedure is as much as $\Theta(n \log M)$, though. To accelerate it we try to detect situations when there is a long snippet near the beginning of the sequence, and apply Lemma 7 and Lemma 8 to quickly process all snippets on its left.

▶ **Lemma 7.** *Given a sequence of extended snippets $s_1 s_2 \ldots s_i$ such that $|s_i| \geq 2 \sum_{j<i} |s_j|$, we can detect an occurrence of p in $s_1 s_2 \ldots s_i$ in time $\mathcal{O}(i)$.*

▶ **Lemma 8.** *Given a sequence of extended snippets $s_1 s_2 \ldots s_i$ such that $|s_i| \geq 2 \sum_{j<i} |s_j|$, we can compute the longest prefix of p which is a suffix of $s_1 s_2 \ldots s_i$ in time $\mathcal{O}(i)$.*

After such modification the algorithm works in linear time, which can be shown by defining a potential function depending just on the lengths of the snippets, see the original paper.

3 Overview of the algorithm

Our goal is to detect an occurrence of a pattern $p[1..M]$ in a given text $t[1..N]$, where p and t are described by a Lempel-Ziv-Welch parse of size m and n, respectively. The difficulty here is rather clear: M might be of order m^2, and hence looking at each possible prefix or suffix of the pattern would imply a quadratic (or higher) total complexity. As most efficient uncompressed pattern algorithms are based on a more or less involved preprocessing concerning all prefixes or suffixes, such quadratic behavior seems difficult to avoid. Nevertheless, we can try to use the following reasoning here: either the pattern is really complicated, and then m is very similar to M, hence we can use the linear compressed pattern matching algorithm sketched in the previous section, or it is in some sense repetitive, and we can hope to speedup the preprocessing by building on this repetitiveness. In this section we give a high level formalization of this intuition.

We will try to process whole codewords at once. To this aim we need the following technical lemma which allows us to compare large chunks of the text (or the pattern) in a single step. It follows from the linear time construction of the so-called *suffix tree of a tree* [17] and constant time LCA queries [2].

▶ **Lemma 9.** *It is possible to preprocess in linear time a LZW parse of a text over an alphabet consisting of integers which can be sorted in linear time so that given any two codewords we can compute their longest common suffix in constant time.*

We defer its proof to Section 6 as it is not really necessary to understand the whole idea. As an obvious corollary, given two codewords we can check if the shorter is a suffix of the longer in constant time.

In the very beginning we reverse both the pattern and the text. This is necessary because the above lemma tells how to compute the longest common suffix, and we would actually like to compute the longest common prefix. The only way we will access the characters of both the pattern and the text is either through computing the longest common prefix of two reversed codewords, or retrieving a specified character of a reversed codeword (which can be performed in constant time using level ancestor queries), hence the input can be safely reversed without worrying that it will make working with it more complicated. We call those reversed codewords *blocks*. Note that all suffixes of a block are valid blocks as well.

We start with classifying all possible patterns into two types. Note that this classification depends on both m (size of the compressed pattern) and n (size of the compressed texts) which might seem a little unintuitive.

▶ **Definition 10.** A *kernel* of the pattern is any (uncompressed) substring of length $n+m$ such that its border is less than $\frac{n+m}{2}$. A kernel decomposition of the pattern is its prefix with period at most $\frac{n+m}{2}$ followed by a kernel.

Note that the distance between two occurrences of such substring must be at least $\frac{n+m}{2}$, and hence a kernel occurs at most $2m$ times in the pattern and $2n$ times in the text. It might happen that there is no kernel, or in other words all relatively short fragments are highly repetitive. In such case the whole pattern turns out to be highly repetitive.

▶ **Lemma 11.** *The pattern either has a kernel decomposition or its period is at most* $\frac{n+m}{2}$. *Moreover, those two situations can be distinguished in linear time, and if a decomposition exists it can be found with the same complexity.*

Proof. We start with decompressing the prefix of length $n+m$. If its period d is at least $\frac{n+m}{2}$, we can return it as a kernel. Otherwise we compute the longest common prefix of the

pattern and the pattern shifted by d characters (or, in other words, we compute how far the period extends in the whole pattern). This can be performed using at most $2(n+m)$ queries described in Lemma 9 (note that being able to check if one block is a prefix of another would be enough to get a linear total complexity here, as we can first identify the longest prefix consisting of whole blocks in both words and then inspect the remaining at most $\min(n,m)$ characters naively). If d is the period of the whole pattern, we are done. Otherwise we identified a substring s of length $n+m$ followed by a character a such that the period of s is at most $d \leq \frac{n+m-1}{2}$ but the period of sa is different (larger). We remove the first character of s and get s'. Let d' be the period of $s'a$. If $d' \geq \frac{n+m}{2}$, $s'a$ is a kernel. Otherwise $d, d' \leq \frac{n+m-1}{2}$ are both periods of s', and hence by Lemma 1 they are both multiplies of the period of s'. Let b be the character such that d is a period of $s'b$ (note that $a \neq b$). Because d is a period of $s'b$, $s'[|s'|+1-d] = b$. Similarly, because d' is a period of $s'a$, $s'[|s'|+1-d'] = a$. Hence $s'[|s'|+1-d] \neq s'[|s'|+1-d']$, and because $(|s'|+1-d)-(|s'|+1-d')$ is a multiple of the period of s' we get a contradiction. Note that the prefix before $s'a$ is periodic with period $d \leq \frac{n+m}{2}$ by the construction. ◀

If the pattern turns out to be repetitive, we try to apply the algorithm described in the preliminaries. The intuition is that while we required a certain preprocessing of the whole pattern, when its period is d it is enough to preprocess just its prefix of length $\mathcal{O}(d)$. This intuition is formalized in Section 4. If the pattern has a kernel, we use it to identify $\mathcal{O}(n)$ potential occurrences, which we then manage to verify efficiently. The verification uses a similar idea to the one from Lemma 9 but unfortunately it turns out that we need to somehow compress the pattern during the verification as to keep the running time linear. The details of this part are given in Section 5.

4 Detecting occurrence of a periodic pattern

If the pattern is periodic, we would like to somehow use this periodicity so that we do not have to preprocess the whole pattern (i.e., build the suffix tree, LCA structure, compute the borders of all prefixes and suffixes, and so on). It seems reasonable that preprocessing just the first few repetitions of the period should be enough. More precisely, we will decompress a sufficiently long prefix of p and compute some of its occurrences inside the text. To compute those occurrences we apply a fairly simple modification of Levered-pattern-matching (see [12]) called Lazy-levered-pattern-matching. The pseudocode of the modified version can be seen below.

First observe that both Lemma 5 and Lemma 7 can be modified in a straightforward way so that we get the leftmost occurrence, if any exists. The original procedure quits as soon as it detects that the pattern occurs. We would like it to proceed so that we get more than one occurrence, though. A naive solution would be to simply continue, but then the following situation could happen: both ℓ and $|s_k|$ are very close to m, the pattern occurs both in the very beginning of $p[1\,..\,\ell]s_k$ and somewhere close to the boundary between the two parts, and the longest suffix of the concatenation which is a prefix of the pattern is very short. Then we would detect just the first occurrence, and for some reasons that will be clear in the proof of Lemma 15 this is not enough. Hence whenever there is an occurrence in the concatenation, we skip just the first half of $p[1\,..\,\ell]$ and continue, see lines 11-14. This is the only change in the algorithm.

While Lazy-levered-pattern-matching is not capable of generating all occurrences in some cases, it will always detect a lot of them, in a certain sense. This is formalized in the following lemma.

Algorithm 1 LAZY-LEVERED-PATTERN-MATCHING$(s_1, s_2, \ldots, s_n)$

1: $\ell \leftarrow$ longest prefix of p ending s_1 ▷ **Lemma 3**
2: $k \leftarrow 2$
3: **while** $k \leq n$ and $\ell + \sum_{i=k}^{n} |s_i| \geq M$ **do**
4: choose $t \geq k$ minimizing $|s_k| + |s_{k+1}| + \ldots + |s_{t-1}| - \frac{|s_t|}{2}$
5: **if** $\ell + |s_k| + |s_{k+1}| + \ldots + |s_{t-1}| \leq \frac{|s_t|}{2}$ **then**
6: output the first occurrence of p in $p[1\,..\,\ell]s_k s_{k+1} \ldots s_t$, if any ▷ **Lemma 7**
7: $\ell \leftarrow$ longest prefix of p ending $p[1\,..\,\ell]s_k s_{k+1} \ldots s_t$ ▷ **Lemma 8**
8: $k \leftarrow t + 1$
9: **else**
10: output the first occurrence of p in $p[1\,..\,\ell]s_k$, if any ▷ **Lemma 5**
11: **if** p occurs in $p[1\,..\,\ell]s_k$ **then**
12: $\ell \leftarrow$ longest prefix of p ending $p[\lceil \frac{\ell}{2} \rceil\,..\,\ell]$
13: **continue**
14: **end if**
15: **if** $p[1\,..\,\ell]s_k$ is a prefix of p **then**
16: $\ell \leftarrow \ell + |s_k|$
17: $k \leftarrow k + 1$
18: **continue**
19: **end if**
20: $b \leftarrow$ longest long border of $p[1\,..\,\ell]$ s.t. $p[1\,..\,b]s_k$ is a prefix of p ▷ **Lemma 6**
21: **if** b is undefined **then**
22: $\ell \leftarrow$ longest prefix of p ending $p[\lceil \frac{\ell}{2} \rceil\,..\,\ell]$
23: **continue**
24: **end if**
25: $\ell \leftarrow b + |s_k|$
26: $k \leftarrow k + 1$
27: **end if**
28: **end while**

▶ **Lemma 12.** *If the pattern of length M occurs starting at the i-th character,* LAZY-LEVERED-PATTERN-MATCHING *detects at least one occurrence starting at the j-th character for some $j \in \{i - \frac{M}{2}, i - \frac{M}{2} + 1, \ldots, i\}$.*

Proof. There are just two places where we can lose a potential occurrence: line 7 and 12. More precisely, it is possible that we output an occurrence and then skip a few others. We would like to prove that the occurrences we skip are quite close to the occurrences we output. We consider the two problematic lines separately.

line 7 s_t is a lever, so $\ell + |s_1| + \ldots + |s_t| \leq \frac{3}{2}M$. Hence the distance between any two occurrences of the pattern inside $p[1\,..\,\ell]s_k s_{k+1} \ldots s_t$ is at most $\frac{M}{2}$. We output the first of them, and so can safely ignore the remaining ones.

line 12 If there is an occurrence, we remove the first half of $p[1\,..\,\ell]$ and might skip some other occurrences starting there. If the first occurrence starts later, we will not skip anything. Otherwise we output the first occurrence starting in $p[1\,..\,\frac{\ell}{2}]$, and if there is any other occurrence starting there, their distance is at most $\frac{\ell}{2} \leq \frac{M}{2}$, hence we can safely ignore the latter.

◀

▶ **Lemma 13.** Lazy-levered-pattern-matching *can be implemented to work in time* $\mathcal{O}(n)$ *and use* $\mathcal{O}(M)$ *additional memory.*

Proof. The proof is almost the same as in [12]. The only difference as far as the running time is concerned is line 12. By removing the first half of $p[1\,..\,\ell]$ we either decrease the current potential by 1 or create a lever and thus can amortize the constant time used to locate the first occurrence of the pattern inside $p[1\,..\,\ell]s_k$. ◀

Note that Lazy-levered-pattern-matching works with a sequence of snippets. By first applying the preprocessing mentioned in the preliminaries we can use it to compute a small set which approximates all occurrences in a compressed text.

▶ **Lemma 14.** Lazy-levered-pattern-matching *can be used to compute a set* S *of* $\mathcal{O}(n)$ *occurrences of an uncompressed pattern of length* $M \geq n$ *in a compressed text such that whenever there is an occurrence starting at the* i*-th character,* S *contains* j *from* $\{i - \frac{M}{2}, i - \frac{M}{2} + 1, \ldots, i\}$.

Proof. As mentioned in the preliminaries, we can reduce compressed pattern matching to pattern matching in a sequence of snippets in linear time. Because $M \geq n$, the preprocessing does not produce any occurrences yet. Then we apply Lazy-levered-pattern-matching. Because its running time is linear by Lemma 13, it cannot find more than $\mathcal{O}(n + M)$ occurrences. A closer look at the analysis shows that the number of occurrences produced can be bounded by the potentials of all sequences created during the initial preprocessing phase, which as shown in [12] is at most $\mathcal{O}(n)$. ◀

Now it turns out that if the pattern is compressed but highly periodic, the occurrences found in linear time by the above lemma applied to a sufficiently long prefix of p are enough to detect an occurrence of the whole pattern.

▶ **Lemma 15.** *Fully compressed pattern matching can be solved in linear time if the period of the pattern is at most* $\frac{n+m}{2}$*. Furthermore, given a set of* r *potential occurrences we can verify all of them in* $\mathcal{O}(n + m + r)$ *time.*

Proof. We build the shortest prefix $p[1\,..\,\alpha d]$ of the pattern such that $\alpha d \geq n + m$, where $d \leq \frac{n+m}{2}$ is the period of the whole pattern. Observe that $\alpha d \leq \frac{3}{2}(n + m)$ and hence we can afford to store this prefix in an uncompressed form. By Lemma 14 we construct a set S of $\mathcal{O}(n)$ occurrences of $p[1\,..\,\alpha d]$ such that for any other occurrence starting at the i-th character there exists $j \in S$ such that $0 \leq i - j \leq \frac{\alpha d}{2} \leq \frac{3}{2}(n + m)$. We partition the elements in S according to their remainders modulo d so that $S_t = \{j \in S : j \equiv t \pmod{d}\}$ and consider each S_t separately. Note that we can easily ensure that its elements are sorted by either applying radix sort to the whole S or simply observing that Lazy-levered-pattern-matching generate the occurrences from left to right.

We split S_t into maximal groups of consecutive elements $x_1 < x_2 < \ldots < x_k$ such that $x_{i+1} \leq x_i + \frac{\alpha d}{2}$, which clearly can be performed in linear time with a single left-to-right sweep. Each such group actually corresponds to a fragment starting at the x_1-th character and ending at the $(x_k + \alpha d - 1)$-th character which is a power of $p[1\,..\,d]$. This is almost enough to detect an occurrence of the whole pattern. If the fragment is sufficiently long, we get an occurrence. In some cases this is not enough to detect the occurrence because we might be required to extend the period to the right as to make sufficient space for the whole pattern. Fortunately, it is impossible to repeat $p[1\,..\,d]$ more than $\frac{3}{2}\alpha$ times starting at the x_k character, as otherwise we would have another $x_{k+1} \in S_t$ which we might have used

to extend the group. Hence to compute how far the period extends it would be enough to align $p[1\,..\,\alpha d]p[1\,..\,\frac{\alpha d}{2}]$ starting at the x_k character and compute the first mismatch with the text. We can assume that all suffixes of $p[1\,..\,\alpha d]$ are blocks with just a linear increase in the problem size, and hence we can apply Lemma 9 to preprocess the input so that each such alignment can be processed in time proportional to the number of block in the corresponding fragment of the text. To finish the proof, note that any single block in the text will be processed at most twice. Otherwise we would have two groups ending at the x_k-th and $x'_{k'}$-th characters such that $|x_k + \alpha d - (x'_{k'} + \alpha d)| \leq \frac{\alpha d}{2}$ and that would mean that one of those groups is not maximal. After computing how far the period extends after each group, we only have to check a simple arithmetic conditions to find out if the pattern occurs starting at the corresponding x_1.

To verify a set of r potential occurrences, we construct the groups and compute how far the period extends after each of them as above. Then for each potential occurrence starting at the b_i-th character we lookup the corresponding $S_{b_i \bmod d}$ and find the rightmost group such that $x_1 \leq b_i$. We can verify an occurrence by looking up how far the period extends after the x_k-th character and checking a simple arithmetic condition. To get the claimed time bound, observe that we do not have to perform the lookup separately for each possible occurrence. By first splitting them according to their remainders modulo d and sorting all x_1 and b_i in linear time using radix sort consisting of two passes we get a linear total complexity. ◀

5 Using kernel to accelerate pattern matching

We start with computing all occurrences of the kernel in both the pattern and the text. Because the kernel is long and aperiodic, there are no more than $2m$ of the former and $2n$ of the latter. The question is if we are able to detect all those occurrences efficiently. It turns out that because the kernel is aperiodic, Lazy-levered-pattern-matching can be (again) used for the task. More formally, we have the following lemma.

▶ **Lemma 16.** Lazy-levered-pattern-matching *can be used to compute in* $\mathcal{O}(n + M)$ *time all occurrences of an aperiodic pattern of length* $M \geq n$ *in a compressed text.*

Proof. By Lemma 14 we can construct in linear time a set of occurrences such that any other occurrence is quite close to one of them. But the pattern is aperiodic, so if it occurs at positions i and j with $|i - j| \leq \frac{M}{2}$, then in fact $i = j$. Hence the set contains all occurrences. ◀

We apply the above lemma to find the occurrences of the kernel in both the pattern and the text. Each occurrence of the kernel in the text gives us a possible candidate for an occurrence of the whole pattern (for example by aligning it with the first occurrence of the kernel in the pattern). Hence we have just a linear number of candidates to verify. Still, the verification is not trivial. An obvious approach would be to repeat a computation similar to the one from Lemma 11 for each candidate. This would be too slow, though, as it might turn out that some blocks from the pattern are inspected multiple times. We require a slightly more sophisticated approach.

Using (any) kernel decomposition of the pattern we represent it as $p = p_1p_2p_3$, where the period of p_1 is at most $\frac{n+m}{2}$, and p_2 is a kernel. We start with locating all occurrences of p_2p_3 in the text. It turns out that because p_2 is aperiodic, there cannot be too many of them. Hence we can afford to generate all such occurrences and then verify if any of them is preceded by p_1 as follows:

Algorithm 2 PREF($T[1..|T|]$)

```
1: PREF[1] = 0, s ← 1
2: for i = 2, 3, ..., |T| do
3:     k ← i − s + 1
4:     r ← s + PREF[s] − 1
5:     if r < i then
6:         PREF[i] = NAIVE-SCAN(i, 1)
7:         if PREF[i] > 0 then
8:             s ← i
9:         end if
10:    else if PREF[k] + k < PREF[s] then
11:        PREF[i] ← PREF[k]
12:    else
13:        x ← NAIVE-SCAN(r + 1, r − i + 2)
14:        PREF[i] ← r − i + 1 + x
15:        s ← i
16:    end if
17: end for
18: PREF[1] = |T|
```

1. if $|p_1| \geq n$ then we can directly apply Lemma 15,
2. if $|p_1| < n$ then take the prefix of p_1p_2 consisting of the first $n+m$ letters. Depending on whether this prefix is periodic with the period at most $\frac{n+m}{2}$ or aperiodic, we can apply Lemma 15 or Lemma 16.

The most involved part is computing all occurrences of p_2p_3. To find them we construct a new string $T = p_2p_3\$t[1..N]$ by concatenating the suffix of the pattern and the text. For this new string we compute the values of the prefix function defined in the following way:

$$\text{PREF}[i] = \max\{j : T[k] = T[i+k-1] \text{ for all } k = 1, 2, \ldots, j\}$$

Of course we cannot afford to compute PREF[i] for all possible $N+M$ values of i. Fortunately, PREF[i] $\geq |p_2|$ iff p_2 occurs in T starting at the i-th character. Because $|p_2| = n+m$ and p_2 is aperiodic, there are no more than $2\frac{N+M}{n+m} \leq n+m$ such values of i. We aim to compute PREF[i] just for those i. First lets take a look at the relatively well-known algorithm which computes all PREF[i] for all i, which can be found in the classic stringology book by Crochemore and Rytter [7]. We state its code for the sake of completeness. NAIVE-SCAN(x, y) performs a naive scanning of the input starting at the x-th and y-th characters and returns the first mismatch, if any. PREF uses this procedure in a clever way as to reuse already processed parts of the input and keep the total running time linear. The complexity is linear because the value of $s + \text{PREF}[s]$ cannot decrease nor exceed $|T|$, and whenever it increases we are able to pay for the time spent in NAIVE-SCAN using the difference between the new and the old value.

We will transform this algorithm so that it computes only PREF[i] such that the kernel occurs starting at the i-th character. We call such positions i *interesting*. The first problem we encounter is that we need a constant time access to any PREF[i] and cannot afford to allocate a table of length $|T|$. This can be easily overcome.

▶ **Lemma 17.** *A table* PREF *such that* PREF$[i] > 0$ *iff the kernel occurs starting at the* i*-th character and any entry can be accessed in constant time can be implemented in space and after preprocessing not exceeding the compressed size of* T*, which is* $\mathcal{O}(n+m)$*.*

Proof. Observe that any two occurrences of the kernel cannot be too close. More precisely, their distance must be at least $\frac{n+m}{2}$. We split the whole T into disjoint fragments of size $\frac{n+m}{2}$. There are no more than $2(n+m)$ of them and there is at most one occurrence in each of them. Hence we can implement the table by allocating an array of size $2(n+m)$ with each entry storing at most one element. ◀

We modify line 2 so that it iterates only through interesting values of i. Note that whenever we access some PREF$[j]$ inside, j is either i, s or $k = i - s + 1$. In the first two cases it is clear that the corresponding positions are interesting so we can access the corresponding value using Lemma 17. The third case is not that obvious, though. It might happen that k is not interesting and we will get PREF$[k] = 0$ instead of the true value. If $r \geq i + |p_2| - 1$ then because p_2 occurs at i, it occurs at k as well, and so k is interesting. Otherwise we cannot access the true value of PREF$[k]$, so we start a naive scan by calling Naive-scan$(i + |p_2|, |p_2| + 1)$ (we can start at the $|p_2| + 1$-th character because p_2 occurs at i). After the scanning we set $s \leftarrow i$. Note that because $r < i + |p_2| - 1$, this increases the current value of $s +$ PREF$[s]$, and we can use the increase to amortize the scanning.

We still have to show how to modify Naive-scan. Clearly we cannot afford to perform the comparisons character by character. By the increasing $s +$ PREF$[s]$ argument, any single character from the text is inspected at most once by accessing $T[x]$ (we call it a left side access). It might be inspected multiple times by accessing $T[y]$, though (which we call a right side access). We would like to perform the comparisons block by block using Lemma 9. After a single query we skip at least one block. If we skip a block responsible for the left side access, we can clearly afford to pay for the comparison. We need to somehow amortize the situation when we skip a block responsible for the right side access. For this we will iteratively compress the input (this is similar to the idea used in [10] with the exception that we work with PREF instead of the failure function). More formally, consider the sequence of blocks describing p_2p_3. First note that no further blocks from T will be responsible for a right side access because of the unique \$ character. Whenever some two neighboring blocks b_1, b_2 from this prefix occur in the same block b' from the text, we would like to glue them, i.e., replace by a single block. We cannot be sure that there exists a block corresponding to their concatenation, but because we know where it occurs in b' we can extract (in constant time, by using the level ancestor data structure [3] to preprocess the whole code trie) a block for which the concatenation is a prefix. We will perform such replacement whenever possible. Unfortunately, after such replacement we face a new problem: p_2p_3 is represented as a concatenation of prefixes of blocks instead of whole blocks. Nevertheless, we can still apply Lemma 9 to compute the longest common prefix of two prefixes of blocks $b[1..i]$ and $b'[1..i']$ by first computing the longest common prefix of b and b', and decreasing it if it exceeds $\min(i, i')$. More formally, we store a *block cover* of p_2p_3.

▶ **Definition 18.** A block cover of a word w is a sequence $b_1[1\,..\,i_1], b_2[1\,..\,i_2], \ldots, b_k[1\,..\,i_k]$ of prefixes of blocks such that their concatenation is equal to w.

This definition is illustrated on Figure 1. Obviously, the initial partition of p_2p_3 into blocks is a valid block cover. If during the execution of Naive-scan we find out that two neighboring elements $b_j[1\,..\,i_1], b_{j+1}[1\,..\,i_{j+1}]$ of the current cover occur in some other longer block b, we replace them with the corresponding prefix of b', see Figure 2.

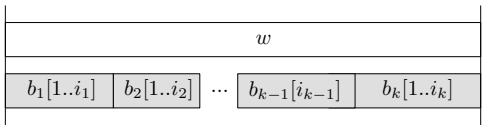


Figure 1 A block cover of w.

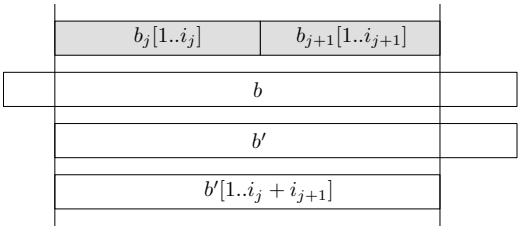


Figure 2 Compressing the current block cover.

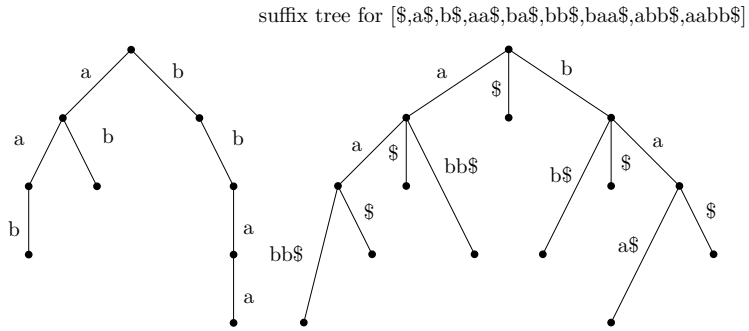


Figure 3 A trie (on the left) and its suffix tree (on the right).

We store all $b_j[1\,..\,i_j]$ on a doubly linked list and update its element accordingly after each replacement. The final required step is to show how we can quickly access in line 13 the block corresponding to $r - i + 2$. We clearly are allowed to spend just constant time there. We keep a pointer to the block covering the r-th character of the pattern. Whenever we need to access the block covering the $(r - i + 2)$-th character, we simply move the pointer to the left, and whenever the current longest match extends, we move the pointer to the right. We cannot move to the left more time than we move to the right, and the latter can be bounded by the number of blocks in the whole p_1p_2 if we replace the neighboring blocks whenever it is possible.

▶ **Lemma 19.** *Fully compressed pattern matching can be solved in linear time if we are given the kernel of the pattern.*

▶ **Theorem 20.** *Fully LZW-compressed pattern matching for strings over a polynomial size integer alphabet can be solved in optimal linear time assuming the word RAM model.*

6 LZW parse preprocessing

The goal of this section is to prove Lemma 9. We aim to preprocess the codewords trie so that given any two codewords, we can compute their longest common suffix in constant time. The *suffix tree of a tree* A, where A is a tree with edges labeled with single characters, is defined as the compressed trie containing $s_A(v)\$$ for all $v \in A$, where $s_A(v)$ is the string constructed by concatenating the labels of all edges on the v-to-root path in A, see Figure 3. This has been first used by Kosaraju [15], who developed a $\mathcal{O}(|A| \log |A|)$ time construction algorithm, where $|A|$ is the number of nodes of A. The complexity has been then improved by Breslauer [6] to just $\mathcal{O}(|A| \log |\Sigma|)$ and by Shibuya [17] to linear for integer alphabets.

We build the suffix tree of the codeword trie T in linear time [17]. As a result we also get for any node v of the input trie the node of the suffix tree corresponding to $s_T(v)\$$.

Now assume that we would like to compute the longest common suffix of two codewords corresponding to nodes u and v in the input trie. In other words, we would like to compute the longest common prefix of $s_T(u)$ and $s_T(v)$. This can be found in constant time after a linear time preprocessing by retrieving the lowest common ancestor of their corresponding nodes in the suffix tree [2].

References

1. Amihood Amir, Gary Benson, and Martin Farach. Let sleeping files lie: pattern matching in z-compressed files. In *SODA '94: Proceedings of the fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 705–714, Philadelphia, PA, USA, 1994. Society for Industrial and Applied Mathematics.
2. Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In *LATIN '00: Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, pages 88–94, London, UK, 2000. Springer-Verlag.
3. Michael A. Bender and Martín Farach-Colton. The level ancestor problem simplified. *Theor. Comput. Sci.*, 321(1):5–12, 2004.
4. Robert S. Boyer and J. Strother Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.
5. Dany Breslauer. Saving comparisons in the Crochemore-Perrin string matching algorithm. In *In Proc. of 1st European Symp. on Algorithms*, pages 61–72, 1995.
6. Dany Breslauer. The suffix tree of a tree and minimizing sequential transducers. In *Proceedings of the 7th Annual Symposium on Combinatorial Pattern Matching*, CPM '96, pages 116–129, London, UK, 1996. Springer-Verlag.
7. Maxime Crochemore and Wojciech Rytter. *Jewels of stringology.* World Scientific, 2002.
8. Zvi Galil. String matching in real time. *J. ACM*, 28(1):134–149, 1981.
9. Zvi Galil and Joel Seiferas. Time-space-optimal string matching (preliminary report). In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 106–113, New York, NY, USA, 1981. ACM.
10. Leszek Gasieniec, Marek Karpinski, Wojciech Plandowski, and Wojciech Rytter. Efficient algorithms for Lempel-Ziv encoding (extended abstract). In *SWAT*, pages 392–403, 1996.
11. Leszek Gasieniec and Wojciech Rytter. Almost optimal fully LZW-compressed pattern matching. In *DCC '99: Proceedings of the Conference on Data Compression*, page 316, Washington, DC, USA, 1999. IEEE Computer Society.
12. Pawel Gawrychowski. Optimal pattern matching in LZW compressed strings. In *SODA '11: Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete algorithms*, pages 362–372, 2011.
13. Donald E. Knuth, James H. Morris, Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(2):323–350, 1977.
14. S. Rao Kosaraju. Pattern matching in compressed texts. In *Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 349–362, London, UK, 1995. Springer-Verlag.
15. S.R. Kosaraju. Efficient tree pattern matching. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:178–183, 1989.
16. James H. Morris, Jr. and Vaughan R. Pratt. A linear pattern-matching algorithm. Technical Report 40, University of California, Berkeley, 1970.
17. Tetsuo Shibuya. Constructing the suffix tree of a tree with a large alphabet. In *Proceedings of the 10th International Symposium on Algorithms and Computation*, ISAAC '99, pages 225–236, London, UK, 1999. Springer-Verlag.

Variable time amplitude amplification and quantum algorithms for linear algebra problems*

Andris Ambainis[1]

1 **Faculty of Computing, University of Latvia, Raina bulv. 19, Riga, LV-1586, Latvia,**
ambainis@lu.lv

Abstract

Quantum amplitude amplification is a method of increasing a success probability of an algorithm from a small $\epsilon > 0$ to $\Theta(1)$ with less repetitions than classically. In this paper, we generalize quantum amplitude amplification to the case when parts of the algorithm that is being amplified stop at different times.

We then apply the new variable time amplitude amplification to give two new quantum algorithms for linear algebra problems. Our first algorithm is an improvement of Harrow et al. algorithm for solving systems of linear equations. We improve the running time of the algorithm from $O(\kappa^2 \log N)$ to $O(\kappa \log^3 \kappa \log N)$ where κ is the condition number of the system of equations. Our second algorithm tests whether a matrix A is singular or far-from-singular, faster then the previously known algorithms.

1998 ACM Subject Classification F.1.2 Modes of computation, F2.1 Numerical algorithms and problems

Keywords and phrases quantum computing, quantum algorithms, amplitude amplification, linear equations

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.636

1 Introduction

Large systems of linear equations arise in many situations and faster algorithms for solving them are of great interest. For this reason, it would be very interesting to have a quantum algorithms for this problem.

However, there are some substantial difficulties with designing such an algorithm. First, if we have a system of linear equations $Ax = b$ with N equations and N unknowns, the coefficient matrix A is of size N^2. If a quantum algorithm accesses all or most of coefficients in A, it would require time $\Omega(N^2)$. We could allow query access to the coefficient matrix A (similarly to Grover's algorithm [12] and other quantum query algorithms) but then we run into a second problem. The quantum algorithm still has to output the solution vector x. Since the solution vector x consists of values for N variables, this requires time $\Omega(N)$.

This argument suggests that quantum speedup for this problem can be at most polynomial (because classical algorithms for systems of linear equations run in time $O(N^\omega)$) where $\omega = 2.37...$ is the matrix multiplication constant.

* Supported by ESF project 1DP/1.1.1.2.0/09/APIA/VIAA/044, FP7 Marie Curie Grant PIRG02-GA-2007-224886 and FP7 FET-Open project QCS.

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 636–647

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

Recently, Harrow, Hassidim and Lloyd [13] discovered a surprising quantum algorithm that allows to bypass the limitations described above and to "solve" systems of linear equations in time $O(\log^c N)$ - in an unconventional sense. Instead of outputting the solution vector x in a classical form, their algorithm generates the quantum state $|x\rangle = \sum_{i=1}^{N} x_i|i\rangle$ with the coefficients x_i being equal to the values of variables in the solution $x = (x_1, x_2, \ldots, x_N)$ of the system $Ax = b$.

HHL algorithm has been quite controversial. On one hand, one cannot read the values $x_1, \ldots, x_N$ from the quantum state $|x\rangle = \sum_{i=1}^{N} x_i|i\rangle$. Even to estimate them, one would have to produce many copies of $|x\rangle$, increasing the running time of the quantum algorithm many times.

On the other hand, the state $|x\rangle = \sum_{i=1}^{N} x_i|i\rangle$ can be used to estimate expressions of the form $\sum_i c_i x_i$ which depend on all x_i simultaneously. Classically, it is intuitively unlikely that one would be able to estimate expressions of this form without solving the system and finding $x_1, \ldots, x_N$ - which requires time $\Theta(N^c)$. This intuition is substantiated by the fact that estimating such expressions is BQP-complete [13]. Thus a classical algorithm for evaluating expressions of the form $\sum_i c_i x_i$ in time $O(\log^c N)$ does not exist - unless P=BQP.

Another context in which the state $|x\rangle$ would be useful is if we wanted to test whether the solutions of two systems of linear equations $Ax = b$ and $A'x' = b'$ were close one to another. In this case, we could generate the solution states $|x\rangle$ and $|x'\rangle$ for both systems and then compare them using the SWAP-test [8]. (For example, we might be interested in testing whether the stationary distributions of two Markov chains are close one to another [13]. Then, each stationary distribution can be described as a solution to a system of linear equations.)

Besides providing the output as a quantum state $|x\rangle$, another weakness of the HHL algorithm is the dependence of its running time on parameters other than the size of the system of equations N. In particular, its running time depends on κ, the condition number of matrix A. The condition number is defined as the ratio between the largest and the smallest singular value of A: $\kappa = \max_{i,j} \frac{|\mu_i|}{|\mu_j|}$ where μ_i are the singular values of A.

When condition number κ is taken into account, the running time of the HHL quantum algorithm is $O(\kappa^2 \log N)$. Thus, the speedup achieved by the HHL algorithm is exponential, as long as $\kappa = O(\log^c N)$. However, systems of linear equations with a polylogarithmic condition number are quite rare. It is much more common for a system to have a condition number that scales as $\Theta(N)$ or $\Theta(N^c)$. (We present some examples in section 4.5.) For this reason, we think that it is important to improve the dependence of the HHL algorithm on κ.

In this paper, we present a better quantum algorithm of solving systems of linear equations in the sense of HHL, with the running time $O(\kappa \log^3 \kappa \log N)$. It would be desirable to have even better dependence on κ but our algorithm is probably close to being optimal. Harrow et al. [13] show that, unless $BQP = PSPACE$, time of $\Omega(\kappa^{1-o(1)})$ is necessary for generating the state $|x\rangle$ that describes the solution of the system.

Our second result is a quantum algorithm for testing whether a matrix A is singular, under a promise that A is either singular or far from being singular. (Here, “far from singular" means that all singular values are at least ϵ.) Under this assumption, we design a quantum algorithm that runs in time[1] $\tilde{O}(\frac{s(A)}{\max(\sqrt{k},1)})$ where k is the number of singular

[1] Here, $\tilde{O}$ notation ignores logarithmic factors.

values of A that are equal to 0 and

$$s(A) = \sqrt{\sum_{i=1}^{N} \frac{1}{\min^2(\rho_i, \epsilon)}}$$

where ρ_i are all the singular values of A.

Both of our results use a new tool, the *variable-time quantum amplitude amplification* which allows to amplify the success probability of quantum algorithms in which some branches of the computation stop earlier than other branches. The conventional amplitude amplification [7] would wait for all branches to stop - possibly resulting in a substantial inefficiency. Our new algorithm amplifies the success probability in multiple stages and takes advantage of the parts of computation which stop earlier.

The variable-time amplitude amplification is a generalization of variable time search of Ambainis [2]. Variable time search was a generalization of Grover's algorithm to the setting when queries to different items take different time. In this paper, we improve variable time search to deal with the more general setting of amplitude amplification.

We then apply the new variable-time amplitude amplification to design the quantum algorithms for solving systems of linear equations and testing singularity. We expect that the variable time amplitude amplification will be useful for building other quantum algorithms, as well.

Related work After this work was completed, Belovs [4] discovered another algorithm for testing the singularity. Belovs' algorithm achieves similar running time, using a different method (span programs) than our work (variable time eigenvalue estimation).

2 Methods and subroutines

Throughout the paper, we use two well known quantum algorithms: *eigenvalue estimation* and *amplitude amplification*.

Eigenvalue estimation. Quantum eigenvalue estimation [15] is a quantum algorithm that, given a Hamiltonian H (in form of a black box that allows by apply H for a time T that we choose) and its eigenstate $|\psi\rangle : H|\psi\rangle = \lambda|\psi\rangle$, outputs an unchanged eigenstate $|\psi\rangle$, together with an estimate $\tilde{\lambda}$ for the eigenvalue λ.

We assume that $0 \leq \lambda \leq 1$. The standard version of eigenvalue estimation [14, p. 118] performs the unitary $U = e^{-iH}$ up to 2^k times and outputs $x \in \{0, \frac{\pi}{2^k}, \frac{2\pi}{2^k}, \ldots, \frac{(2^k-1)\pi}{2^k}\}$ with probability

$$p(x) = \frac{1}{2^{2k}} \frac{\sin^2 2^k(\lambda - x)}{\sin^2(\lambda - x)} \tag{1}$$

(equation (7.1.30) from [14]).

According to Theorem 7.1.5 in [14], if $\lambda \in [\frac{m}{2^k}, \frac{m+1}{2^k}]$, then the probability of outputting one of the two closest estimates ($\frac{m}{2^k}$ and $\frac{m+1}{2^k}$)) is at least $\frac{8}{\pi^2}$. We can increase this probability to at least $1-\epsilon$ by repeating the eigenvalue estimation algorithm $O(\log \frac{1}{\epsilon})$ times and taking the majority of answers.

Amplitude amplification. Another tool that we repeatedly use is quantum amplitude amplification [7]. Quantum amplitude amplification takes an algorithm $\mathcal{A}$ that succeeds with a small probability ϵ and transforms it into an algorithm $\mathcal{A}'$ that succeeds a probability $2/3$ (or $1 - o(1)$).

Classically, increasing the success probability from ϵ to $2/3$ requires repeating $\mathcal{A}$ $\Theta(\frac{1}{\epsilon})$ times. Quantumly, amplitude amplification allows to do that with just $O(\frac{1}{\sqrt{\epsilon}})$ repetitions of

$\mathcal{A}$. The algorithm $\mathcal{A}$ whose success probability is being increased can be either a classical algorithm or a quantum algorithm.

3 Variable time amplitude amplification

Our first result is a generalization of the amplitude amplification. Consider a quantum algorithm $\mathcal{A}$ which may stop at one of several times $t_1, \ldots, t_m$. (In the case of singularity-testing or systems of linear equations, these times correspond to m runs of eigenvalue estimation with increasing precision and increasing number of steps.) To indicate the outcome, $\mathcal{A}$ has an extra register O with 3 possible values: 0, 1 and 2. 1 indicates the outcome that should be amplified. 0 indicates that the computation has stopped at this branch but did not result in the desired outcome 1. 2 indicates that the computation at this branch has not stopped yet.

Let p_i be the probability of the algorithm stopping at time t_i (with either the outcome 0 or outcome 1). The average stopping time of $\mathcal{A}$ (the l_2 average) is

$$T_{av} = \sqrt{\sum_i p_i t_i^2}.$$

T_{max} denotes the maximum possible running time of the algorithm (which is equal to t_m). Let

$$\alpha_{good}|1\rangle_O|\psi_{good}\rangle + \alpha_{bad}|0\rangle_O|\psi_{bad}\rangle$$

be the algorithm's output state after all branches of the computation have stopped. Our goal is to obtain $|\psi_{good}\rangle$ with a high probability. Let $p_{succ} = |\alpha_{good}|^2$ be the probability of obtaining this state via algorithm $\mathcal{A}$.

Our main result is

▶ Theorem 1. We can construct a quantum algorithm $\mathcal{A}'$ invoking $\mathcal{A}$ several times, for total time

$$O\left(T_{max}\sqrt{\log T_{max}} + \frac{T_{av}}{\sqrt{p_{succ}}}\log^{1.5} T_{max}\right)$$

that produces a state $\alpha|1\rangle \otimes |\psi_{good}\rangle + \beta|0\rangle \otimes |\psi'\rangle$ with probability $|\alpha|^2 \geq 1/2$ as the output[2].

Proof. The proof is given in the full version of the paper [3]. ◀

By repeating $\mathcal{A}'$ $O(\log \frac{1}{\epsilon})$ times, we can obtain $|\psi_{good}\rangle$ with a probability at least $1-\epsilon$.

In contrast to our algorithm, the usual amplitude amplification [7] would run for time $O(\frac{T_{max}}{\sqrt{p_{succ}}})$. Our algorithm $\mathcal{A}'$ provides an improvement whenever T_{av} is substantially smaller than T_{max}.

Our algorithm $\mathcal{A}'$ is optimal, up to the factor of $\log^c T_{max}$. If the algorithm $\mathcal{A}$ has just one stopping time $T = T_{av} = T_{max}$, then amplitude amplification cannot be performed with fewer than $O(\frac{T}{\sqrt{p_{succ}}})$ steps. Thus, the term of $\frac{T_{av}}{\sqrt{p_{succ}}}$ is necessary.

If we would like to algorithm $\mathcal{A}'$ to be exact (to produce an output state that is exactly $|\psi_{good}\rangle$, conditional on the first bit being $|1\rangle$), the term T_{max} is also necessary because, in some branch of computation, $\mathcal{A}$ can run for T_{max} steps and $\mathcal{A}'$ needs the part of $|\psi_{good}\rangle$ that comes from this branch. If $\mathcal{A}'$ only has to produce an approximation of $|\psi_{good}\rangle$, a better result is possible.

[2] The first bit of the output state indicates whether we have the desired state $|\psi_{good}\rangle$ or not. Since $|\alpha|^2 \geq 1/2$, we get $|\psi_{good}\rangle$ with probability at least $1/2$.

▶ Theorem 2. Let $\epsilon > 0$ be a constant. We can construct a quantum algorithm $\mathcal{A}'$ invoking $\mathcal{A}$ several times, for total time

$$O\left(\frac{T_{av}}{\sqrt{p_{succ}}}\log^{1.5}\max\left(T_{av},\frac{1}{p_{succ}}\right)\right)$$

that produces a state $\alpha|1\rangle\otimes|\psi'_{good}\rangle+\beta|0\rangle\otimes|\psi'\rangle$ with $|\alpha|^2\geq 1/2$ and $\|\psi_{good}-\psi'_{good}\|\leq\epsilon$ as the output.

Theorem 2 is a straightforward consequence of Theorem 1. We observe that the probability of an algorithm $\mathcal{A}$ running for more than $T_0=\frac{T_{av}}{\sqrt{\delta p_{succ}}}$ steps is at most δp_{succ}. (Otherwise, we would have $T_{av}^2>\delta p_{succ}T_0^2=T_{av}$.)

We set $\delta=(\epsilon/2)^2$ and $T_0=\frac{T_{av}}{\epsilon\sqrt{p_{succ}/2}}$ and take a quantum algorithm $\mathcal{A}_1$ that runs $\mathcal{A}$ but stops after T_0 steps. Then, the output state of $\mathcal{A}_1$ is $|\psi'_{good}\rangle$ with[3] $\|\psi_{good}-\psi'_{good}\|\leq\epsilon$ and T_{max} for the new algorithm $\mathcal{A}_1$ is equal to T_0. Theorem 2 now follows by applying Theorem 1 to $\mathcal{A}_1$.

4 Quantum algorithms for linear algebra problems

4.1 Preliminaries

We will consider two problems: testing whether a matrix A is singular and "solving" systems of linear equations $Ax=b$ in the sense of [13].

Similarly to [13], we assume that the matrix A is Hermitian. This assumption is without a loss of generality. For singularity testing, if A is not Hermitian, we can replace it by

$$A'=\begin{pmatrix}0 & A\\ A^\dagger & 0\end{pmatrix}, \tag{2}$$

where 0 denotes the all-zero matrix of the appropriate size. Then, A' is singular if and only if A is singular.

For systems of linear equations, we can replace $Ax=b$ by $A'y=b'$ where $b'=\begin{pmatrix}0\\ b\end{pmatrix}$. The solution of this system is

$$y=\begin{pmatrix}x\\ 0\end{pmatrix}$$

which is essentially equivalent to x.

For both algorithms, the matrix A can be given in one of the following forms:

1. A black box implementing A (for Hermitian A) as a Hamiltonian;
2. A black box answering queries about the values of A, in one of the following two forms:
   a. (for dense matrices) given i,j, the black box returns a_{ij};
   b. (for sparse matrices) given i, the black box returns a list of all values in the i^{th} row (or i^{th} column) that are non-zero.

[3] Removing the part of $|\psi_{good}\rangle$ that corresponds to $\mathcal{A}$ running for more than T_0 steps results in an unnormalized state $|\psi''_{good}\rangle$ with $\|\psi_{good}-\psi''_{good}\|\leq\epsilon/2$. Normalizing $|\psi''_{good}\rangle$ results in a normalized state $|\psi'_{good}\rangle$ with $\|\psi''_{good}-\psi'_{good}\|\leq\epsilon/2$ and $\|\psi_{good}-\psi'_{good}\|\leq\epsilon$.

The second case reduces to the first one, because, given a black box that answers queries about values a_{ij}, we can build a black box implementing A, by using one of methods for simulating black-box Hamiltonians. In the sparse case, to simulate the Hamiltonian A for time T, it is sufficient to use the query black box for A $O((T\log N)^{1+o(1)})$ times [5, 10]. In the dense case, the quantum-walk based methods by Childs [9] give an $O(C(A)T)$ query simulation of the Hamiltonian A for time T, with a somewhat complicated dependence of $C(A)$ on the matrix A.

For the rest of this paper, we assume that A is given via a Hamiltonian. We assume that the evolution of the Hamiltonian A for time T can be simulated in time $C(A)\min(T,1)$, for some $C(A)$ (as in the simulation by [9]). (Using a simulation method that works in time $O(C(A)T^{1+o(1)})$ (as in [5, 10]) is also possible, with a corresponding increase in the running times of our algorithms.)

4.2 Singularity testing

We consider the problem of testing whether a matrix A is singular. It is known that testing the singularity of an $n\times n$ matrix requires $\Omega(n^2)$ queries in the quantum query model [11].

However, better quantum algorithms may be possible for restricted cases of the singularity problem. A natural restriction is to consider the case when the matrix A is either singular or far from being singular.

Namely, we consider the testing whether A is singular with a promise that $\|A\|\leq 1$ and one of the following two is true:

- A is singular;
- All singular values of A are at least ϵ.

We will refer to this problem as ϵ-*Singularity*.

Let $\rho_1(A),\ldots,\rho_N(A)$ be the singular values of a matrix A. Let

$$s(A)=\sqrt{\sum_{i=1}^{N}\frac{1}{\min^2(\rho_i,\epsilon)}}.$$

If A is not Hermitian, we replace it by a Hermitian, as described in section 4.1. If $\rho_1(A),\ldots,\rho_N(A)$ are the singular values of a matrix A, then the eigenvalues of A' are $\pm\rho_1(A),\ldots,\pm\rho_N(A)$.

▶ Theorem 3. There is an algorithm $\mathcal{A}$ for ϵ-singularity that runs in time $O(C(A)s(A)\log^{1.5}s(A)\log N)$ if A is non-singular and time

$$O\left(\frac{C(A)s(A)\log^{1.5}s(A)\log N}{\sqrt{k}}\right)$$

if A has $k>0$ singular values that are equal to 0.

Proof. We apply variable time amplitude amplification to Algorithm 1.

To analyze this algorithm, we first observe that receiving the second part of a completely mixed state as an input is equivalent to receiving the N-dimensional completely mixed state ρ_N as the input. The completely mixed state can be written as a mixture of eigenvectors $|v_i\rangle$ of A with equal coefficients:

$$\rho_N=\sum_{i=1}^{N}\frac{1}{N}|v_i\rangle\langle v_i|.$$

Input: an $N \times N$ matrix A.

1. With probability $\frac{1}{2N}$ output "non-singular" and stop.
2. Prepare a bipartite state

$$\sum_{i=1}^{N} \frac{1}{\sqrt{N}} |i\rangle \otimes |i\rangle.$$

3. Let $k = 1$.
4. While $k \leq \lceil \frac{2}{\epsilon} \rceil$, do:
   a. On the second register, apply eigenvalue estimation for A with parameters chosen so that, with probability at least $1 - \frac{1}{N^2}$, the estimate is within $\frac{1}{2^k}$ of being correct.
   b. If the obtained estimate is at least $\epsilon + \frac{1}{2^k}$, output "fail" and stop.
   c. If the obtained estimate is at most $\epsilon - \frac{1}{2^k}$, output "singular" and stop.

Algorithm 1 Algorithm for singularity testing.

If the input to eigenvalue estimation was $|v_i\rangle$, the eigenvalue estimation loop would stop after $O(\frac{1}{\rho_i} \log N)$ steps, with a high probability. Therefore, the l_2-average stopping time would be $T_{av} = O(\frac{s(A) \log N}{\sqrt{N}})$.

Let $\mathcal{A}$ be the algorithm obtained by applying Theorem 2 to Algorithm 1. If A has k singular values that are equal to 0, then the success probability of Algorithm 1 is $p_{succ} = \frac{k+0.5}{N}$ and the running time of $\mathcal{A}$ is

$$O\left(\frac{T_{av}}{\sqrt{p_{succ}}} \log^{1.5} \max(T_{av}, p_{succ})\right) = O\left(\frac{s(A) \log^{1.5} s(A) \log N}{\sqrt{k}}\right).$$

Conditional on the algorithm succeeding, the probability of the correct answer “singular" is $\frac{k}{k+0.5} \geq \frac{2}{3}$.

If A has no singular value equal to 0, then the success probability of Algorithm 1 is $p_{succ} = \frac{1}{2N} + O(\frac{1}{N^2})$, with the $O(\frac{1}{N^2})$ term coming from the possibility that eigenvalue estimation may output an incorrect estimate with probability (at most $\frac{1}{N^2}$). The running time of $\mathcal{A}$ is

$$O\left(\frac{T_{av}}{\sqrt{p_{succ}}} \log^{1.5} \max(T_{av}, p_{succ})\right) = O\left(s(A) \log^{1.5} s(A) \log N\right).$$

Conditional on the algorithm succeeding, the probability of the correct answer “non-singular" is $\frac{\frac{1}{2N}}{p_{succ}} = 1 - o(1)$. ◀

4.3 Systems of linear equations

We consider solving a system of linear equations $Ax = b$ where $A = (a_{ij})_{i,j \in [N]}$, $x = (x_i)_{i \in [N]}$, $b = (b_i)_{i \in [N]}$. As before, we assume that A is Hermitian.

Let $|v_i\rangle$ be the eigenvectors of A and λ_i be their eigenvalues. Similarly to [13], we assume that all λ_i satisfy $\frac{1}{\kappa} \leq |\lambda_i| \leq 1$ for some known κ. We can then transform the state $|b\rangle = \sum_{i=1}^{n} b_i |i\rangle$ into $|x\rangle = \sum_{i=1}^{n} x_i |i\rangle$ as follows:

1. If, in terms of eigenvectors $|v_i\rangle$ of A, we have $|b\rangle = \sum_i c_i |v_i\rangle$, then $|x\rangle = \sum_i \frac{c_i}{\lambda_i} |v_i\rangle$.
2. By eigenvalue estimation, we can create the state $|b'\rangle = \sum_i c_i |v_i\rangle |\tilde{\lambda}_i\rangle$ where $\tilde{\lambda}_i$ are the estimates of the true eigenvalues.

3. We then create the state

$$|b''\rangle = \sum_i c_i |v_i\rangle |\tilde{\lambda}_i\rangle \left(\frac{1}{\kappa\tilde{\lambda}_i}|1\rangle + \sqrt{1 - \frac{1}{\kappa^2\tilde{\lambda}^2}}|0\rangle \right). \quad (3)$$

   Conditional on the last bit being 1, the rest of state is $\sum_i \frac{c_i}{\tilde{\lambda}_i}|v_i\rangle|\tilde{\lambda}_i\rangle$ which can be turned into an approximation of $|x\rangle$ by running eigenvalue estimation in reverse and uncomputing $\tilde{\lambda}_i$.
4. We then amplify the part of state which has the last qubit equal to 1 (using amplitude amplification) and obtain a good approximation of $|x\rangle$ with a high probability.

▶ Theorem 4. [13] Let $Ax = b$ be a system of linear equations. Then, we can generate $|\psi\rangle$ satisfying $\||\psi\rangle - |x\rangle\| \leq \epsilon$ where $|x\rangle = \sum_{i=1}^n x_i|i\rangle$ in time $O(\frac{C(A)\kappa^2}{\epsilon}\log N)$.

The main term in the running time, κ^2 is generated as a product of two κ's. First, for $\||\psi\rangle - |x\rangle\| \leq \epsilon$, it suffices that the estimates $\tilde{\lambda}_i$ satisfy $|\lambda_i - \tilde{\lambda}_i| = O(\epsilon\tilde{\lambda}_i)$. Since $\lambda_i = \Omega(1/\kappa)$, this means $|\lambda_i - \tilde{\lambda}_i| = O(\frac{\epsilon}{\kappa})$. To estimate λ_i within error $O(\frac{\epsilon}{\kappa})$, we need to run H for time $O(\frac{\kappa}{\epsilon})$. Second, for amplitude amplification, we may need to repeat the algorithm generating $|b''\rangle$ $O(\kappa)$ times - resulting in the total running time $O(\kappa^2/\epsilon)$.

For eigenvalue estimation, the worst case is when all of most of λ_i are small (of order $\Theta(1/\kappa)$). Then, $|\lambda_i - \tilde{\lambda}_i| = \Theta(\frac{\epsilon}{\kappa})$ and eigenvalue estimation with the right precision indeed requires time $\Theta(\frac{\kappa}{\epsilon})$.

For amplitude amplification, the worst case is if most or all of λ_i are large (constant). Then, the coefficients $\frac{1}{\kappa\tilde{\lambda}_i}$ can be of order $\Theta(1/\kappa)$ and $\Theta(\kappa)$ repetitions are required for amplitude amplification.

We now observe that the two $\Theta(\kappa)$'s appear in the opposite cases. One of them appears when λ_i is small ($\lambda_i \approx \kappa$) but the other appears when λ_i is large ($\lambda_i \approx 1$).

If all eigenvalues are of roughly similar magnitude (e.g., $\lambda \in [a, 2a]$ for some a), the running time becomes $O(\kappa/\epsilon)$ because we can do eigenvalue estimation in time to error ϵa in $O(1/a\epsilon)$ and, for amplitude amplification, it suffices to repeat the generation of $|b''\rangle$ $O(\kappa a)$ times (since the amplitude of 1 in the last qubit of $|b'\rangle$ is at least $\frac{1}{\kappa a}$ for every v_i). Thus, the running time is

$$O\left(\frac{1}{a\epsilon}\right) \cdot O(\kappa a) = O\left(\frac{\kappa}{\epsilon}\right).$$

The problem is to achieve a similar running time in the general case (when the eigenvalues λ_i can range from κ to 1).

To do that, we run eigenvalue estimation several times. Each time, we double the precision and double the running time (as in Algorithm 1 for singularity testing). This gives a quantum algorithm in which different branches of computation stop at different times. By applying our variable-time amplitude amplification to this quantum algorithm, we get

▶ Theorem 5. Let $Ax = b$ be a system of linear equations. Then, we can generate $|\psi\rangle$ satisfying $\||\psi\rangle - |x\rangle\| \leq \epsilon$ in time

$$O\left(\frac{C(A)\kappa\log^3\frac{\kappa}{\epsilon}}{\epsilon^3}\log^2\frac{1}{\epsilon}\right).$$

For more details, we refer the reader to the full version of the paper [3].

4.4 Algorithm of Theorem 5

In this subsection, we describe the algorithm of Theorem 5. For its analysis, we refer the reader to the full version of this paper [3].

For our algorithm, we need a version of eigenvalue estimation that is guaranteed to output exactly the same estimate with a high probability. This can be achieved by running the standard eigenvalue estimation (described in section 2) k_{uniq} times and takes the most frequent answer x_{maj}.

▶ Lemma 1. For $k_{uniq} = O(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon})$, we have

1. If $|\lambda - x| \leq \frac{1-\epsilon}{2^{n+1}}$, then $Pr[x_{maj} = x] \geq 1 - \epsilon$.
2. If $\lambda \in [x + \frac{1-\epsilon}{2^{n+1}}, x + \frac{1+\epsilon}{2^{n+1}}]$, then $Pr[x_{maj} \in \{x, x+1\}] \geq 1 - \epsilon$.

Proof. Omitted. ◀

We refer to this algorithm as **UniqueEst**$(H, 2^n, \epsilon)$.

When we use **UniqueEst** as a subroutine in algorithm 3, we need the answer to be unique (as in the first case) and not one of two high-probability answers (as in the second case). To deal with that, we will replace H with $H + \frac{\delta\pi}{2^n} I$ for a randomly chosen $\delta \in [0, 1]$. The eigenvalue becomes $\lambda' = \lambda + \frac{\delta\pi}{2^n}$ and, with probability $1 - \epsilon$,

$$\lambda' \in \left[\frac{x - \frac{1-\epsilon}{2}}{2^n}\pi, \frac{x + \frac{1-\epsilon}{2}}{2^n}\pi \right]$$

for some integer x. This allows to achieve the first case for all eigenvalues, except a small random fraction of them.

We now show that Theorem 1 implies our main result, Theorem 5. We start by describing a variable running time Algorithm 2. This algorithm uses the following registers:

- The input register I which holds the input state $|x\rangle$ (and is also used for the output state);
- The outcome register O, with basis states $|0\rangle, |1\rangle$ and $|2\rangle$ (as described in the setup for variable-time amplitude amplification);
- The step register S, with basis states $|1\rangle$, $|2\rangle$, ..., $|2m\rangle$ (to prevent interference between various branches of computation).
- The estimation register E, which is used for eigenvalue estimation (which is a subroutine for our algorithm).

$\mathcal{H}_I$, $\mathcal{H}_O$, $\mathcal{H}_S$ and $\mathcal{H}_E$ denote the Hilbert spaces of the respective registers.

From now on, we refer to ϵ appearing in Theorem 5 as ϵ_{final}. ϵ without a subscript is an error parameter for subroutines of algorithm 2 (which we will choose at the end of the proof so that the overall error in the output state is at most ϵ_{final}).

Our main algorithm is Algorithm 3 which consists of applying variable-time amplitude amplification to Algorithm 2.

We claim that, conditional on the output register being $|1\rangle_O$, the output state of Algorithm 2 is close to

$$|\psi_{ideal}\rangle = \sum_i \alpha_i |v_i\rangle_I \otimes \left(\frac{1}{\kappa\lambda_i} |1\rangle_O \otimes |2j_i\rangle_S \right). \tag{5}$$

Variable-time amplitude amplification then generates a state that is close to $\frac{|\psi_{ideal}\rangle}{\||\psi_{ideal}\rangle\|}$. Fourier transform in the last step of algorithm 3 then effectively erases the S register. Conditional

Input: parameters $x_1, \ldots, x_m \in [0,1]$, Hamiltonian H.

1. Initialize O to $|2\rangle$, S to $|1\rangle$ and E to $|0\rangle$. Set $j = 1$.
2. Let $m = \lceil \log_2 \frac{\kappa}{\epsilon} \rceil$.
3. Repeat until $j > m$:

   Stage j:

   a. Let $H' = H + \frac{x_j \pi}{2^j} I$. Using the registers I and S, run **UniqueEst**$(H', 2^j, \epsilon)$. Let λ' be the estimate output by **UniqueEst** and let $\lambda = \lambda' - \frac{x_j \pi}{2^j}$.

   b. If $\epsilon\lambda > \frac{1}{2^{j+1}}$, perform the transformation

$$|2\rangle_O \otimes |1\rangle_S \rightarrow \frac{1}{\kappa\lambda}|1\rangle_O \otimes |2j\rangle_S + \sqrt{1 - \frac{1}{(\kappa\lambda)^2}}|0\rangle_O \otimes |2j\rangle_S. \quad (4)$$

   c. Run **UniqueEst** in reverse, to erase the intermediate information.

   d. Check if the register E is in the correct initial state $|0\rangle_E$. If not, apply $|2\rangle_O \otimes |1\rangle_S \rightarrow |0\rangle_O \otimes |2j+1\rangle_S$ on the outcome register O.

   e. If the outcome register O is in the state $|2\rangle$, increase j by 1 and go to step 2.

Algorithm 2 State generation algorithm

Input: Hamiltonian H.

1. Generate uniformly random $x_1, \ldots, x_m \in [0,1]$.
2. Apply variable-time amplitude amplification to Algorithm 2, with H and $x_1, \ldots, x_m$ as the input.
3. Apply a transformation mapping $|2j\rangle_S \rightarrow |j\rangle_S$ to the S register. After that, apply Fourier transform F_m to the S register and measure. If the result is 0, output the state in the I register. Otherwise, stop without outputting a quantum state.

Algorithm 3 Main algorithm

on S being in $|0\rangle_S$ after the Fourier transform, the algorithm's output state is close to our desired output state $\frac{|x\rangle}{\||x\|\|}$, where

$$|x\rangle = \sum_i \alpha_i |v_i\rangle_I.$$

Finally, performing Fourier transform and measuring produces $|0\rangle_S$ with probability $1/m$. Because of that, the success probability of algorithm 3 needs to be amplified. This adds a factor of $O(\sqrt{m})$ to the running time, if we would like to obtain the result state with probability $\Omega(1)$ and a factor of $O(\sqrt{m}\log\frac{1}{\epsilon})$ if we would like to obtain it with probability at least $1-\epsilon$.

4.5 Examples of systems of linear equations

The Harrow-Hassidim-Lloyd algorithm achieves the biggest speedup when the condition number κ is small. If κ is polylogarithmic in N, then $O(\kappa^2 \log^c N) = O(\log^{c'} N)$. The Harrow-Hassidim-Lloyd algorithm then achieves an exponential speedup compared to the classical algorithms which run in time that is polynomial in N.

In this case, the additional advantage provided by our algorithm is small. However, systems of linear equations for which $\kappa = O(\log^c N)$ are quite rare. (We have looked at possible applications of the HHL algorithms and it was difficult to find natural examples of

systems where $\kappa = O(\log^c N)$.) We illustrate this with several natural examples of systems of linear equations.

Example 1: Assume that we have a system of equations $Ax = b$ in which A and b are random (for example, each entry is an i.i.d random variable which takes values +1 and -1 with probability 1/2 each).

With a high probability, the biggest singular value of A is of the order $\Theta(\sqrt{N})$ and the smallest singular value of A is of the order $\Theta(1/\sqrt{N})$ [16]. Hence, $\kappa = \Theta(N)$.

Thus, the running time of the HHL algorithm would be

$$O(\kappa^2 C(A) \log^c N) = O(N^2 C(A) \log^c N).$$

(For arbitrary A, with query access to A, $C(A) = O(N)$ [9]. This would give the overall running time of $O(N^3 \log^c N)$ - worse than classical algorithms for solving systems of linear equations.)

Theorem 5 provides an improvement of the running time to

$$O(\kappa \log^3 \kappa C(A) \log^c N) = O(NC(A) \log^c N).$$

Example 2: Consider a d-dimensional grid of size $\sqrt[d]{N} \times \sqrt[d]{N} \times \ldots \times \sqrt[d]{N}$, consisting of locations $(a_1, \ldots, a_d)$, $a_i \in \{1, 2, \ldots, \sqrt[d]{N}\}$. Let L be the Laplacian of this grid, defined by

$$L_{(a_1,\ldots,a_d),(b_1,\ldots,b_d)} = \begin{cases} 2d & \text{if } (a_1, \ldots, a_d) = (b_1, \ldots, b_d) \\ -1 & \text{if } a_i = b_i \pm 1 \text{ for one } i \text{ and } a_i = b_i \text{ for all other } i \\ 0 & \text{otherwise} \end{cases}$$

Consider a system of linear equations of the form $Lx = b$.

We can express $L = L_1 + L_2 + \ldots + L_d$ where

$$(L_i)_{(a_1,\ldots,a_d),(b_1,\ldots,b_d)} = \begin{cases} 2 & \text{if } (a_1, \ldots, a_d) = (b_1, \ldots, b_d) \\ -1 & \text{if } a_i = b_i \pm 1 \text{ and } a_j = b_j \text{ for all } j \neq i \\ 0 & \text{otherwise} \end{cases}$$

For simplicity, we assume that the grid has periodic boundary conditions (i.e., location $\sqrt[d]{N} + 1$ equals location 1). Then, the eigenvalues of L_i are $\lambda_j = 2 - 2\cos\frac{j\pi}{\sqrt[d]{N}}$, for $j = 0, 1, \ldots, \sqrt[d]{N} - 1$. Since $\cos x \approx 1 - \frac{x^2}{2}$ for small x, the smallest non-zero eigenvalue is

$$2 - 2\cos\frac{\pi}{\sqrt[d]{N}} \approx 2\left(\frac{\pi}{\sqrt[d]{N}}\right)^2 = \Theta\left(\frac{1}{N^{2/d}}\right).$$

The largest eigenvalue is upper bounded by 4.

The eigenvalues of L are of the form $\lambda_{j_1} + \lambda_{j_2} + \ldots + \lambda_{j_d}$ where λ_{j_i} are the eigenvalues of L_i. Hence, the smallest non-zero eigenvalue is $\Theta(\frac{1}{N^{2/d}})$ while the largest eigenvalue is $\Theta(d)$. The condition number is $O(dN^{2/d})$.

For $d = \log N$, the condition number would be of the order $O(\log N)$ and both HHL and our algorithm would run in polylogarithmic time. However, a more interesting case would be $d = 2$ or $d = 3$, since this would correspond to a discretization of actual physical processes in 2 or 3 dimensions. Then, $\kappa = O(N)$ (for $d = 2$) or $\kappa = O(N^{2/3})$ (for $d = 3$).

In this case, the HHL algorithm would run in time $\tilde{O}(N^2)$ or $\tilde{O}(N^{4/3})$. Our algorithm would improve this to $\tilde{O}(N)$ or $\tilde{O}(N^{2/3})$. (Since the Laplacian L is sparse, the overhead due to simulating Hamiltonian L is small.)

References

1 S. Aaronson, A. Ambainis, Quantum search of spatial regions. *Theory of Computing*, 1:47-79, 2005. Also quant-ph/0303041.

2 A. Ambainis. Quantum search with variable times. *Theory of Computing Systems*, 47(3): 786-807, 2010. Earlier version in STACS'08 and quant-ph/0609188.

3 A. Ambainis. Variable time amplitude amplification and quantum algorithms for linear algebra problems. arXiv:1010.4458.

4 A. Belovs. Span-program-based quantum algorithm for the rank problem. arXiv:1103.0842.

5 D.W. Berry, G. Ahokas, R. Cleve, and B.C. Sanders. Efficient Quantum Algorithms for Simulating Sparse Hamiltonians. *Communication in Mathematical Physics*, 270(2):359-371, 2007. Also arXiv:quant-ph/0508139

6 D. Berry. Quantum algorithms for solving linear differential equations. arXiv:1010.2745.

7 G. Brassard, P. Høyer, M. Mosca, A. Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information Science*, AMS Contemporary Mathematics Series, 305:53-74, 2002. Also quant-ph/0005055.

8 H. Buhrman, R. Cleve, J. Watrous, and R. de Wolf. Quantum fingerprinting. *Physical Review Letters*, 87(16):167902, 2001. Also quant-ph/0102001

9 A. M. Childs. On the relationship between continuous- and discrete-time quantum walk, *Communications in Mathematical Physics*, 294:581-603, 2010. Also arXiv:0810.0312.

10 A. M. Childs, R. Kothari. Simulating sparse Hamiltonians with star decompositions. *Proceedings of TQC 2010*, Lecture Notes in Computer Science 6519:94-103, 2011. Also arXiv:1003.3683.

11 S. Dörn, T. Thierauf. The quantum query complexity of the determinant. *Information Processing Letters*, 109 (6):325-328, 2009.

12 Lov K. Grover. A fast quantum mechanical algorithm for database search. *Proceedings of STOC'96*, pp. 212-219. Also quant-ph/9605043.

13 A. Harrow, A. Hassidim, S. Lloyd, Quantum algorithm for linear systems of equations. *Physical Review Letters*, 15(103):150502, 2009. Also arxiv:0811.3171.

14 P. Kaye, R. Laflamme, M. Mosca. *An Introduction to Quantum Computing.* Cambridge University Press, 2007.

15 M. Mosca, A. Ekert. The Hidden Subgroup Problem and Eigenvalue Estimation on a Quantum Computer. *Proceedings of QCQC'98*, Lecture Notes in Computer Science, 1509: 174-188, 1998. Also quant-ph/9903071.

16 M. Rudelson, R. Vershynin. The Littlewood - Offord problem and invertibility of random matrices. *Advances in Mathematics*, 218:600-633, 2008.

17 J. Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, 1994.

Weak MSO+U over infinite trees*

Mikołaj Bojańczyk[1] and Szymon Toruńczyk[†2]

1 **University of Warsaw**
2 **INRIA and ENS Cachan**

Abstract

We prove that, over infinite trees, satisfiability is decidable for Weak Monadic Second-Order Logic extended by the unbounding quantifier U. We develop an automaton model, prove that it is effectively equivalent to the logic, and that the automaton model has decidable emptiness.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.1 Mathematical Logic

Keywords and phrases Infinite trees, distance automata, MSO+U, profinite words

Digital Object Identifier 10.4230/LIPIcs.STACS.2012.648

1 Introduction

The general topic of this paper is monadic second-order logic extended with the unbounding quantifier. The unbounding quantifier is a kind of set quantifier, which says that a formula $\varphi(X)$ holds for arbitrarily large finite sets X:

$$\mathsf{U}X\varphi(X) \qquad \stackrel{\text{def}}{=} \qquad \bigwedge_{n\in\mathbb{N}} \exists X\ n \leq |X| < \infty \ \wedge\ \varphi(X).$$

The unbounding quantifier was introduced in [1], along with some rudimentary decidability results. The quantifier is part of a research program, which investigates the notion of "regular language" for infinite words and trees. The general theme of the research program is that some features, such as the unbounding quantifier, can be added to monadic second-order logic over infinite objects, while preserving properties one would expect from a regular language. For instance, consider a language L of infinite words. Define a Myhill-Nerode-like equivalence relation $\sim_L$ on finite words:

$$w \sim_L w' \quad \text{if for every finite word } u \text{ and every infinite word } v, \quad uwv \in L \iff uw'v \in L.$$

One can show that if L is defined in monadic second-order logic with the unbounding quantifier (MSO+U), then $\sim_L$ has finitely many equivalence classes. Furthermore, each equivalence class is a regular language of finite words. The research program is discussed in [3].

The expressive power of the logic MSO+U is still not properly understood. It is an open problem whether satisfiability is decidable over infinite words.

So far, research has dealt with fragments of the logic. The paper [4] introduces two classes of automata on infinite words, called ωB- and ωS-automata, and proves that they correspond to fragments of MSO+U with restricted quantifier use. It is not clear if there can be an automaton model for the whole logic MSO+U, as opposed to an automaton model for fragments of the logic. These doubts are based on the paper [12], which proves that

* Both authors have been partially supported by ERC Starting Grant Sosna.
† partially supported by the Polish Ministry of Science grant nr N N206 567840

29th Symposium on Theoretical Aspects of Computer Science (STACS'12).
Editors: Christoph Dürr, Thomas Wilke; pp. 648–660

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


MSO+U can define non-Borel languages of infinite words. This implies that there can be no nondeterministic automaton model for MSO+U that has a Borel acceptance condition, which excludes all known nondeterministic automata models that use counters. One has to keep in mind that the non-Borel result still leaves room for automata; a distant analogy is that parity automata on infinite trees recognize non-Borel sets.

The topological problems described above disappear when one considers *weak* monadic second-order logic (WMSO), where set quantifiers are restricted to finite sets. In countable structures, such as infinite words or trees, formulas of WMSO, even extended with the unbounding quantifier, can only define Borel languages. Over infinite words, and without the unbounding quantifier, WMSO has the same expressive power as MSO, thanks to the McNaughton/Safra determinization theorem. This coincidence fails when the unbounding quantifier is introduced: WMSO+U is strictly less powerful than MSO+U. The crucial advantage of the weak logic is that it supports the classical automaton-logic connection: it admits an automaton model, the max-automaton from [2]. The automaton-logic connection also works for other extensions of WMSO on infinite words, see [5]. The topological complexity of WMSO+U has been studied in [7].

Content of this paper

The goal of this paper is the following theorem:

▶ **Theorem 1.** *Satisfiability is decidable for* WMSO+U *over infinite trees.*

We prove the theorem in three steps.

1. In Section 2, we define a new automaton model for infinite trees, called a nested limsup automaton, which has the same expressive power as WMSO+U, and show effective translations from the logic to the automaton and back again.
2. In Section 3, we define a second new automaton model for infinite trees, called a puzzle, which is more expressive than a nested limsup automaton, and show an effective translation from nested limsup automata to puzzles.
3. In Section 4, we provide a decision procedure for nonemptiness of puzzles.

The proof, especially step 3, is maybe more interesting than the result itself. The general theme is to extend concepts of automata theory from finite sets to infinite sets equipped with compact metric topologies. The details of the proof are deferred to an external appendix [6], which is divided into three parts.

Related work

The automata models studied in this paper work on infinite objects. Two of the models, namely the ωB- and ωS-automata from [4], have natural counterparts working on finite words, called B- and S-automata. These finite word counterparts have recently seen a lot of interest. Instead of defining boolean-valued functions, which accept or rejects words, B- and S-automata define number-valued functions, which map words to numbers. These number-valued functions on finite words have been studied in depth by Colcombet in [9], under the name of regular cost functions. The theory of regular cost functions looks very promising, see [11] and [10] for some developments.

On a technical level, this paper uses profinite words [13] to model the limit behavior of finite words. This approach has been successfully applied in [14], as an alternative for cost

functions in the study of the limitedness problem – B- or S-automata do define boolean-valued functions, which accept or reject profinite words.

Acknowledgement. We are grateful to the anonymous reviewer for his detailed remarks.

2 WMSO+U and Nested Limsup Automata

In Section 2 and 3, when talking about a tree over an alphabet A, we mean a full infinite binary tree with nodes labeled by A. (In Section 4, we switch to edge-labeled graphs.)

WMSO+U. A tree is interpreted as a logical structure, with unary predicates for labels, and two binary predicates for left and right successors. To express properties of this logical structure, we use weak monadic second-order logic, which means that formulas can quantify over nodes and *finite* sets of nodes. We use the convention where first-order variables are denoted x, y, z and set variables are denoted X, Y, Z. Also, we allow the unbounding quantifier U defined in the introduction.

▶ Running example. Consider an alphabet $A = \{a, b, c\}$. Define a *b-factor* in a tree t to be a connected set of nodes with label b. Being a b-factor is definable in WMSO:

$$\text{bfactor}(X) \stackrel{\text{def}}{=} \exists x\ x \in X \wedge \big(\forall z\ z \in X \implies \big(x \leq z \ \wedge\ \forall y\ x \leq y \leq z \implies (y \in X \wedge b(y))\big)\big).$$

Here, $\leq$ denotes the ancestor relation – the transitive reflexive closure of the parent relation – which is definable in WMSO. The running example in this paper is the tree language over A, call it L, which contains a tree if and only if the root has label a, and for every node x:

(a) If x has label a, then its subtree has b-factors of unbounded size.
(b) If x has label b or c, then in its subtree, the size of b-factors is bounded.

The language L is defined by the following formula of WMSO+U

$$(\exists x\ \forall y\ (x \leq y) \wedge a(x)) \quad \wedge \quad \big(\forall x\ a(x) \iff \mathsf{U}X\big((\text{bfactor}(X) \wedge \forall y\ (y \in X \Rightarrow y \geq x)\big)\big).$$

Let L be the set of trees satisfying the property above. What does a tree $t \in L$ look like? Observe first that every b-factor has to be finite, since an infinite b-factor contains finite b-factors of unbounded size, violating condition (b). Also, a node with label b or c cannot have a descendant with label a. This is because a tree with b-factors of bounded size cannot have a subtree with b-factors of unbounded size. It follows that all the nodes with label a form a connected set, call it X, which contains the root. There must be b-factors of unbounded size below every node from X, however every such b-factor must be finite, and have bounded size b-factors in its subtree. It follows that every node from X has at least one child in X, and the size of b-factors with parents in X is unbounded. An example is depicted in Figure 1. In the figure, we distinguish the maximal b-factors and call them $F_1, F_2, \ldots$, because they will get a lot of attention in the later analysis. The language L contains no regular tree, because in a regular tree either b-factors have bounded size, or some b-factor is infinite. In particular, L is not a regular language of infinite trees. Observe that in Figure 1, the only part of the tree that behaves in a non-regular way is the b-factors $F_1, F_2, \ldots$. ◀

2.1 Nested Limsup Automata

In this section, we define an automaton model which, over infinite trees, has the same expressive power as WMSO+U. The automaton is obtained by nesting two types of automata: prefix automata and limsup automata. We begin by defining prefix automata and limsup automata, then we show how they are nested.

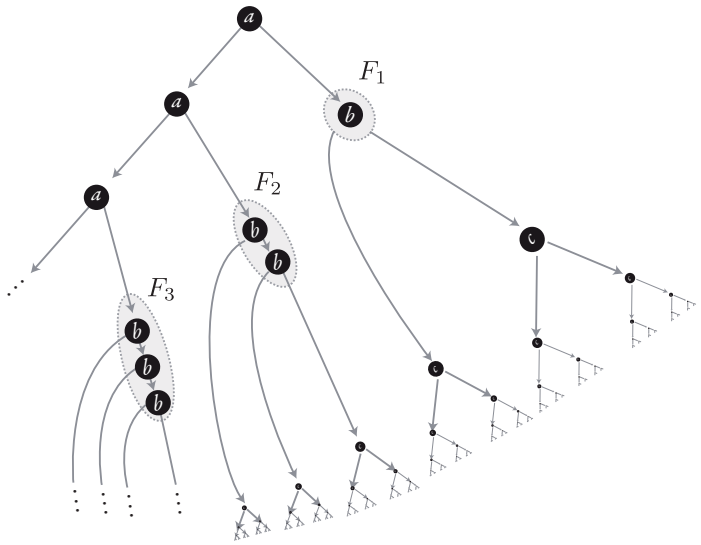


Figure 1 A tree $t \in L$. Every c node has only c descendants.

Prefix automata. A prefix automaton is used to test regular properties of a finite prefix of a tree. Typical languages recognized by this kind of automaton are reachability properties "some node has label a", or "there is an antichain with five labels a". A *prefix* of a tree is an ancestor-closed set of nodes in the tree. A *prefix automaton* is given by the following ingredients:

- An *input alphabet* A.
- A finite set of *states* Q, together with an initial state $q_I \in Q$.
- A *(nondeterministic) transition relation*

$$\delta \subseteq Q \times A \times Q \times Q.$$

- A set of *accepting states* $F \subseteq Q$

The automaton accepts an infinite tree if there is a finite prefix $X \subseteq \{0,1\}^*$ and a run $\rho : X \to Q$, such that ρ respects the transition relation, has the initial state in the root, and all maximal nodes of X have labels in the accepting set F.

A prefix automaton has an existential nature: it tests if there exists a finite prefix with a certain (regular) property. In particular, languages recognized by prefix automata are open sets, under the usual topology over infinite trees.

Atomic limsup automata. We now define a second kind of automaton, called an atomic limsup automaton. A typical language recognized by this kind of automaton is "for every $n \in \mathbb{N}$, there is some path in the tree with at least n labels a". Observe that this typical language is not the same as "there is some path with infinitely many labels a".

The general idea is that the automaton has a counter, which stores natural numbers. The transition function chooses states in a top-down deterministic fashion. The transition function also induces a labeling of edges in the tree by sequences of counter operations. There are two counter operations: increment (written *inc*) and reset (written *reset*). Unlike the model for WMSO+U on infinite words defined in [2], there is no max operation here.

The automaton accepts an input tree if the counter has unbounded values, ranging over nodes in the tree. We give a formal definition below.

An *atomic limsup automaton* is given by the following ingredients:

- An *input alphabet* A.
- A finite set of *states* Q, together with an initial state $q_I \in Q$.
- A *(top-down deterministic) transition function*

$$\delta : Q \times A \to (\{inc, reset\}^* \times Q)^2.$$

Let t be a tree over the input alphabet A. Using the deterministic transition function δ and the initial state in the root, one labels in a unique way the nodes of t by states and the edges of t by sequences in $\{inc, reset\}^*$. Suppose that the counter has value 0 in the root. For any finite path π in t, by reading the operations along the path, we get a counter value. The automaton accepts the tree t if the counter value is unbounded, when ranging over all finite paths in the tree. In other words, the automaton accepts if there are arbitrarily long sequences of increments that are not interrupted by reset.

Nested limsup automata. We now combine the two automata above into a single model, by using nesting. We define nested limsup automata by induction on the *nesting depth*. A nested limsup automaton of nesting depth 1 is either a prefix automaton, or an atomic limsup automaton.

An automaton of nesting depth $k+1$ is defined as follows. Suppose that $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are nested limsup automata of nesting depth k, over a common input alphabet A. Let $\mathcal{B}$ be either a prefix automaton, or an atomic limsup automaton, with input alphabet $\{0,1\}^n$. Then the expression $\mathcal{B}[\mathcal{A}_1, \ldots, \mathcal{A}_n]$ defines a nested limsup automaton. This new automaton has nesting depth $k+1$ and input alphabet A. When does it accept a tree t? Consider the tree $\hat{t}$ over alphabet $\{0,1\}^n$, where the label of a node x is a bit-vector, which has 1 on coordinate $i \in \{1, \ldots, n\}$ if and only if $\mathcal{A}_i$ accepts the subtree of t rooted in x. The automaton $\mathcal{B}[\mathcal{A}_1, \ldots, \mathcal{A}_n]$ accepts t if and only if the automaton $\mathcal{B}$ accepts the tree $\hat{t}$.

Observe that nested limsup automata are closed under complementation – the complement of $\mathcal{A}$ is recognized by an automaton $\mathcal{B}[\mathcal{A}]$, where $\mathcal{B}$ is a prefix automaton checking for 0 at the root.

Like all nested models of automata, nested limsup automata are something of a hybrid, sitting between logical formulas and automata.

▶ Running example. We now present a nested limsup automaton which recognizes the complement of the language L from the running example. Consider first an auxiliary automaton $\mathcal{B}$, a limsup automaton, which increments its counter whenever it sees a b, and resets it whenever it sees a or c. Since a large b-factor must contain a long path, the automaton $\mathcal{B}$ accepts a tree if and only if the tree has b-factors of unbounded size. A tree belongs to the complement of L if and only if the root is not labeled by an a, or if there is some node x, such that:

- The label of x is a, and $\mathcal{B}$ rejects the subree of x; or
- The label of x is either b or c, and $\mathcal{B}$ accepts the subtree of x.

Therefore, the complement of L is recognized by a limsup automaton nested inside a prefix automaton. ◀

2.2 Equivalence

The model of nested limsup automata is designed to be equivalent to WMSO+U, as stated in the following theorem.

▶ **Theorem 2.** *A language of infinite trees is definable in* WMSO+U *if and only if it is recognized by a nested limsup automaton. Translations both ways are effective.*

The proof of this theorem is in part I of the appendix [6]. The proof ideas are based on [2].

Recall that our goal in this paper is to decide satisfiability of WMSO+U. The above theorem reduces the satisfiability problem of WMSO+U to the emptiness problem for nested limsup automata. However, due to the nesting operation, nested limsup automata are still too difficult to solve for emptiness. That is why, in the next section, we present a further reduction, which removes the nesting in a nested limsup automaton.

3 Puzzles

We now turn to the second automaton model in this paper, which is called a puzzle. The name is silly because we do not expect this model to be relevant outside this paper.

3.1 Puzzles, a denested version of nested limsup automata.

The ingredients of a *puzzle* are:

- a finite set Q of *states*
- a finite set C of *counters*
- an input alphabet A
- an *initial state* $q_I \in Q$
- a (nondeterministic) *transition relation*

$$\delta \subseteq Q \times A \times (\{inc, reset, cut\} \times C)^* \times Q)^2$$

- an *unbounding acceptance condition* $q \in Q \mapsto \mathsf{U}_q \subseteq C$, which maps each state q to the set of counters that are called *unbounded in* q.
- a *parity acceptance condition* $q \in Q \mapsto \Omega_q \in \mathbb{N}$, which maps each state to a natural number, called its *parity rank*.

Given an input tree t over the input alphabet, a *run* of the puzzle is an infinite binary tree where the nodes are labeled by states, the root has the initial state, and the edges are labeled by $(\{inc, reset, cut\} \times C)^*$, in a way consistent with the transition relation of the puzzle.

Observe that there is a new counter operation, called *cut*. The idea is that in the acceptance condition, the lim sup operation is only calculated along paths without cut. More formally, for a sequence of counter operations

$$\sigma \in (\{inc, reset, cut\} \times C)^*$$

we define the value of σ on counter c, denoted by $\mathrm{val}(\sigma, c)$, to be the maximal number n, such that some prefix of σ without a cut on counter c has n increments on counter c that are not interrupted by a reset on counter c. For example,

$$\mathrm{val}(\sigma, c) = 2 \qquad \text{for } \sigma = inc(c)cut(d)inc(c)cut(c)inc(d)inc(c)inc(c)inc(c)reset(c).$$

even though there are 3 consecutive increments on c after the cut on c. For a finite path π in a run ρ, we define

$$\mathrm{val}(\rho, \pi, c) \stackrel{\text{def}}{=} \mathrm{val}(\sigma, c) \qquad \text{where } \sigma \text{ is the sequence of edge labels on } \pi.$$

Finally, for a node x in a run ρ, we define

$$\mathrm{val}(\rho, x, c) \stackrel{\text{def}}{=} \sup\{\mathrm{val}(\rho, \pi, c) : \pi \text{ is a finite path originating in } x\ \} \in \mathbb{N} \cup \{\infty\}.$$

A run ρ is *accepting* if on every path, the parity acceptance condition is satisfied, and

$$\mathsf{U}_q = \{c \in C : \mathrm{val}(\rho, x, c) = \infty\} \qquad \text{for every node } x \text{ with state } q. \tag{1}$$

The key differences between a puzzle and a nested limsup automaton are:

- The set of bounded counters is tested in every subtree, as defined in (1);
- The model is not nested, but nondeterministic;
- There is the new *cut* counter operation.

▶ Running example. We define a puzzle which recognizes the language L from the running example (for simplicity, we ignore the condition on the root label). The states Q are q_a, q_b and q_c. There is one counter, call it d. State q_b increments the counter, which corresponds to counting the size of a path in a b-block, while the other states q_a and q_c reset the counter. This behavior is captured by the following set of transitions:

$$\{(q_\sigma, \sigma, (\mathit{reset}(d), q_0), (\mathit{reset}(d), q_1)) : \sigma \in \{a, c\}, q_0, q_1 \in Q\} \cup \\ \{(q_b, b, (\mathit{inc}(d), q_0), (\mathit{inc}(d), q_1)) : q_0, q_1 \in Q\}.$$

In this particular puzzle, the parity acceptance condition plays no role, and all states have accepting parity rank 0. Also, this puzzle does not use the *cut* operation. The key role is played by the unbounding acceptance condition, which is defined by

$$\mathsf{U}_{q_a} = \{d\} \qquad \mathsf{U}_{q_b} = \emptyset \qquad \mathsf{U}_{q_c} = \emptyset.$$

In other words, any node with state q_a in an accepting run must have unbounded values of the counter in its subtree, and every other node must have bounded values of the counter in its subtree. ◀

▶ **Theorem 3.** *For every nested limsup automaton one can compute a puzzle that recognizes the same language.*

The proof of this theorem is in part II of the appendix [6]. The theorem can be interpreted as trading nesting for nondeterminism. From the point of view of deciding emptiness, this is a good trade: nesting is cumbersome for an emptiness algorithm, while nondeterminism is irrelevant.

The converse of Theorem 3 fails: thanks to the parity condition, puzzles recognize non-Borel tree languages, while WMSO+U defines only Borel tree languages. Another reason is shown in the appendix: languages recognized by puzzles are not closed under complements.

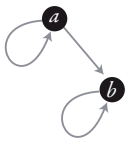
4 Emptiness for puzzles

This section is about the emptiness procedure for puzzles.

▶ **Theorem 4.** *Emptiness is decidable for puzzles.*

The general idea is that even though an accepting run of a puzzle is an infinite object, there should be some way of drawing it in a finite way. This idea works for Büchi automata, because every nonempty Büchi automaton accepts the unfolding of a lasso graph such as:

This idea also works for parity tree automata, because every nonempty parity tree automaton accepts the unfolding of some finite graphs, such as:



Runs as graphs. In the proof of Theorem 4, we will treat a run ρ of a puzzle as an edge-labeled graph G_ρ. The graph G_ρ has the same nodes as ρ. It has no labels on the nodes. An edge in the graph is labeled by the word

$$\sigma \in Q \cdot (\{inc, reset, cut\} \times C)^* \cdot Q,$$

which begins with the state in the source node of the edge, followed by the sequence of counter operations on the edge, and ending with the state in target node of the edge. From now on, when writing ρ, we will refer to the graph G_ρ. The labels on the edges of G_ρ are words over the alphabet

$$\Lambda \stackrel{\text{def}}{=} Q \cup \{inc, reset, cut\} \times C.$$

We fix this alphabet for the rest of this section.

▶ Running example. Recall the tree t from Figure 1. The puzzle in the running example has only one run over any tree, and over t the run is accepting. The part of this run that concerns that b-factor F_n is illustrated in Figure 2. In the rest of this section, we will try to define a limit F_ω. ◀

Factor. A *factor* in a tree is a connected set of nodes. Every factor has a root node, which is the least node in the factor. A *port* in a factor is a node outside the factor that has its parent in the factor. A *root-to-port path* in a factor is a path from the root to some port, seen as a sequence of edges.

Signature. The signature of a (finite or infinite) path in a run is the concatenation of all the labels on that path, which is a word over the alphabet Λ. We use the letters σ or τ to denote signatures of paths.

▶ Running example. In Figure 2, the signature of the rightmost root-to-port path in F_n is

$$\sigma_n \stackrel{\text{def}}{=} (q_b \cdot inc \cdot q_b)^{n-1} \cdot (q_b \cdot inc \cdot q_c).$$ ◀

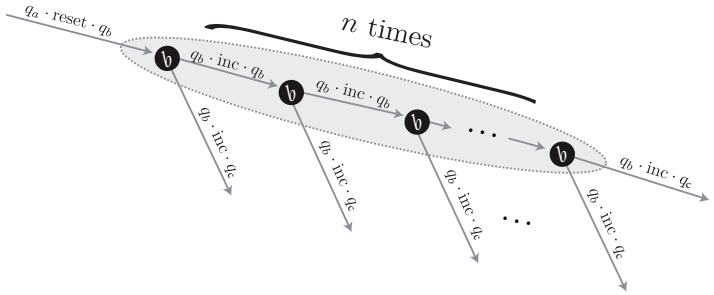


Figure 2 The run of the puzzle inside the b-factor F_n of the tree t from Figure 1.

For signatures of factors, we use multisets, which are sets where the number of occurrences an element can be in $\mathbb{N} \cup \{\infty\}$. Consider a *finite* factor F. The *signature* of the factor is the multiset of path signatures, ranging over root-to-port paths in the factor. All path signatures in this multiset have the same source state, namely the state in the root of the factor. This state is called the root state of a factor signature. It is important that factor signatures describe finite factors. In an infinite factor, it may be the case that a path is not included in root-to-port paths. We use the letter Σ to denote factor signatures.

▶ Running example. The signature of the factor in Figure 2 is the multiset, call it Σ_n, which contains all path signatures $\sigma_1, \ldots, \sigma_{n-1}$ once, and the path signature σ_n twice. ◀

4.1 Limits of signatures

The key technique in this paper is to use limits. We are mainly interested in the limits of signatures, both signatures of paths, and signatures of factors. In this section, we establish the notion of limit that we use. Our approach to limits of path signatures is to treat path signatures as a special case of profinite words. Our approach to limits of factor signatures is to use a variant of Hausdorff distance on multisets of profinite words. The definitions follow.

Profinite words. Consider the following distance on finite words over the alphabet Λ.

$$\mathrm{distance}(\sigma, \tau) = \max\{\frac{1}{2^n} : \text{ some DFA of } n \text{ states accepts } \sigma \text{ but not } \tau\}.$$

It is not difficult to see that this is indeed a distance, even an ultrametric:

$$\mathrm{distance}(\sigma_1, \sigma_2) \leq \max(\mathrm{distance}(\sigma_1, \tau), \mathrm{distance}(\tau, \sigma_2)) \qquad \text{for every } \sigma_1, \sigma_2, \tau \in \Lambda^*.$$

A sequence of words $(\tau_n)_n$ is called *Cauchy* if for every $\varepsilon > 0$ there is some n such that all the words $\tau_n, \tau_{n+1}, \ldots$ lie in a ball of diameter ε.

▶ Running example. Recall the sequence $(\sigma_n)_n$ of signatures of rightmost paths in the factors F_n. This sequence is not Cauchy, because even-numbered words have an even number of increments, and odd-numbered words do not, and evenness can be tested by a DFA of 2 states. However, the sequence $(\sigma_n)_n$ has several Cauchy subsequences, including the sequences $(\sigma_{n!})_n$ and $(\sigma_{n!+1})_n$. ◀

Consider two Cauchy sequences $(\sigma_n)_n$ and $(\tau_n)_n$ to be *equivalent* if

$$\sigma_1, \tau_1, \sigma_2, \tau_2, \ldots$$

is also a Cauchy sequence. This is an equivalence relation, call it $\sim$. An equivalence class of this relation is called a *profinite word* (see [13] for more on profinite words). The set of profinite words is denoted by $\widehat{\Lambda^*}$. We model signatures of paths and their limits by profinite words. Here are the key properties of profinite words that we use:

1. It makes sense to say that a profinite word belongs or does not belong to a regular language $L \subseteq \Lambda^*$. Indeed, if (σ_n) is a Cauchy sequence, then either all but finitely many elements belong to L, or all but finitely many elements do not belong to L. Therefore, it makes sense to say that a Cauchy sequence belongs or does not belong to a regular language. Also this property is preserved when going to an equivalent Cauchy sequence. In particular, it makes sense to say that a profinite word does at least one increment on some counter c, or does at least 4 increments, or begins with state q, because all of these are regular properties.
2. There is a distance on profinite words, namely:

$$\text{distance}((\sigma_n)_n, (\tau_n)_n) = \lim_{n\to\infty} \text{distance}(\sigma_n, \tau_n).$$

   By the triangle inequality, the above limit exists for Cauchy sequences and does not depend on the choice of a sequence in a class of $\sim$. Equipped with this distance, the set of profinite words is a compact metric space. This means that every sequence has a converging subsequence. Also, for every regular language $L \subseteq \Lambda^*$, there is some distance ε such that any two words at distance at most ε either both belong to L, or both do not belong to L.
3. It makes sense to concatenate profinite words. This is because the relation $\sim$ is a congruence with respect to concatenation of sequences:

$$(\sigma_n)_n \sim (\sigma'_n)_n \quad \text{and} \quad (\tau_n)_n \sim (\tau'_n)_n, \qquad \text{implies} \qquad (\sigma_n \cdot \tau_n)_n \sim (\sigma'_n \cdot \tau'_n)_n.$$

▶ Running example. Recall the Cauchy sequence $(\sigma_{n!})_n$. We write σ_ω for the profinite word represented by this sequence. This profinite word begins with letter q_b and ends with letter q_c. Also, for every n, there are more than n increments in σ_ω, which is something that can only happen in a profinite word. ◀

Hausdorff distance on sets. So far, we have defined a compact metric space to model path signatures, namely the set of profinite words over Λ. We now want to do the same thing for multisets of path signatures. Our approach is to use a multiset variant of the Hausdorff distance on sets. We begin by recalling the distance on sets (not multisets), because this definition is easier to digest. A metric on a set A (we are interested in the case when A is the set of profinite words over Λ) can be lifted to a metric on closed subsets of A, using the Hausdorff distance. For two closed subsets $X, Y \subseteq A$, their Hausdorff distance is defined by

$$\text{distance}(X, Y) \stackrel{\text{def}}{=} \max\big(\sup_{x\in X}\inf_{y\in Y} \text{distance}(x, y), \sup_{y\in Y}\inf_{x\in X} \text{distance}(x, y)\big).$$

This is a metric on closed subsets. This definition can be extended to so-called *closed multisets*. As an example, consider multisets of real numbers. Like any finite multiset, the multiset

$$X_n = \{\frac{1}{n}, \frac{1}{n^2}, \ldots, \frac{1}{n^n}\}$$

is closed. The sequence $(X_n)_n$ tends to the (closed) multiset where 0 appears infinitely often.

▶ Running example. Consider the signature Σ_n of the factor F_n. One can prove that the sequence $(\Sigma_{n!})_n$ is Cauchy. Its limit is the multiset, call it Σ_ω, where every σ_n appears once, and every limit of a subsequence of (σ_n) appears infinitely often. Among others, for every $k \in \mathbb{N}$, $\sigma_{\omega+k}$ appears infinitely often. ◀

4.2 Signature graphs

We are now ready to define the key concept of this paper, which is a signature graph. A signature graph is used to represent limits of accepting runs. A signature graph is going to have labeled parallel edges, so it is really a multigraph. When talking about an *edge labeled multigraph*, we mean a directed graph with edges labeled by some alphabet A. The edges form a multiset, so for any label σ and pair of vertices x, y, the number of edges from x to y with label σ may potentially be $0, 1, \ldots$, or countably infinite.

Definition of a signature graph. A *path signature* is a profinite word over Λ that begins with a state (called the *source* state) and ends with a state (called the *target* state). A *factor signature* is a closed multiset of path signatures, which agree on the source state. A *signature graph* is a multigraph with edges labeled by path signatures, subject to the following consistency condition. For every node x, there is some state $q \in Q$, such that all edges entering x have target state q, and all edges leaving x have source state q. This state q is called the *node label* of x, although technically speaking a signature graph supplies only edge labels, and the node labels are derived information.

In a signature graph, the labeling assigns signature paths to individual edges. However, using the monoid structure of path signatures, we can assign a path signature to every finite edge path, by concatenating the labels of the edges in the path.

Fans. Suppose that x is a node in an edge labeled graph. We define the *fan* of x to be the multiset of labels of edges leaving x. If $\mathscr{P}$ is a family of multisets over A, then we say that a graph has *fans in* $\mathscr{P}$ if the fan of each node is in $\mathscr{P}$. In the proof, when dealing with signature graphs, we are interested in two sets $\mathscr{P}$ of factor signatures. The first set, call it

$$\mathscr{P}_{\text{fin}}$$

is the set of factor signatures of finite factors that appear in some run of the puzzle, not necessarily accepting. This set depends on the transitions and counter in the puzzle. It does not depend, however, on the acceptance conditions (boundedness and parity) in the puzzle, because these are only used to distinguish accepting runs. The second set is the closure of the first set, under the Hausdorff distance, in the space of closed multisets of profinite words:

$$\overline{\mathscr{P}_{\text{fin}}}.$$

Limit accepting signature graph. Recall that thanks to the properties of the profinite monoid, it makes sense to say that a path signature does an increment/cut/reset on some counter. We say that a path signature has *value* ω *on counter* c if for every $n \in \mathbb{N}$, the path signature has value at least n on counter c (recall that the value refers to the maximal number of increments, not interrupted by a reset, before the cut). Also, one can ask about the maximum rank, in the parity acceptance condition, of states visited by the limit path signature. For a node x in a signature graph G, define $\mathsf{U}(G, x)$ to be the set of counters U_q, where q is the state in the label of x.

We now present the key definition in the emptiness procedure for puzzles, the definition of a limit accepting signature graph. The idea is that a limit accepting signature graph is the limit of a converging sequence of accepting runs.

A signature graph G with a distinguished *root node* is called *limit accepting* if

1. The root node is labeled by the initial state of the automaton,
2. Every node in G is reachable from the root node,
3. The parity condition is satisfied on every infinite path,
4. For every node x and counter c, counter c belongs to $\mathsf{U}(G, x)$ if and only if
   a. There is an infinite path from x, such that every prefix of the path can be extended to a finite path that does not cut c, and reaches a node whose fan contains an edge with ω on counter c; or
   b. There is an infinite path from x, which does not cut c, resets it finitely often, and increments it infinitely often.

Main technical theorem. To prove Theorem 4, we present a stronger result, Theorem 5, which is the main technical contribution of this paper.

▶ **Theorem 5.** *The following conditions are equivalent.*

1. *There is a limit accepting signature graph with fans in* $\overline{\mathscr{P}_{\text{fin}}}$,
2. *There is a limit accepting signature graph with fans in* $\overline{\mathscr{P}_{\text{fin}}}$ *with finitely many nodes,*
3. *The puzzle has an accepting run.*

Furthermore, given a puzzle one can decide if the conditions hold.

▶ Running example. By Theorem 5, the puzzle from the running example should have a limit accepting signature graph with fans in $\overline{\mathscr{P}_{\text{fin}}}$, with finitely many nodes. Such a graph is illustrated in Figure 3, and has 3 nodes, but infinitely many edges. ◀

The proof of Theorem 5 is in part III of the appendix [6]. A rough sketch is as follows.

- **Implication from 1 to 2**. The key point is that we can design an automaton model, closely resembling alternating automata on graphs, which recognizes limit accepting signature graphs. This automaton model shares the following property with alternating automata on graphs: a nonempty automaton accepts a graph with finitely many nodes.
- **Implication from 2 to 3.** The key point is to get rid of the limits, and replace them by actual finite pieces of runs. The idea is of course to use finite pieces of runs from a sequence approximating the limit, but the implementation of this idea requires some technical effort. We use a notion of bisimulation that is adapted to converging sequences.

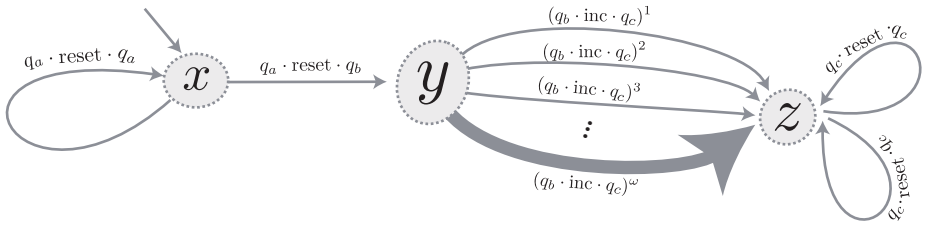


Figure 3 This signature graph represents the accepting run in Figure 2, in the following sense. Node x represents all nodes with label a. The self-loop in x stands for the lefmost path in the graph from Figure 2. Node y represents a limit of the factors F_n. The fan of y is Σ_ω – the limit of the fans $(\Sigma_{n!})_n$. The thick edge stands for infinitely many edges, including infinitely may with label $(q_b \cdot \mathit{inc} \cdot q_c)^\omega$. Node z together with its two self-loops stands for a subtree with infinitely many c's.

- **Implication from 3 to 1.** The key point is to extract limits from an arbitrary accepting run of a puzzle. For this, we use a version of Ramsey's theorem adapted to metric spaces.
- **Decidability.** The key point is to compute a finite representation of the set $\overline{\mathscr{P}_{\mathrm{fin}}}$. In the end, we reduce this to the domination problem for B-automata over finite trees [11].

We believe that the technique of limits of graphs is quite general, and can be applied to other automaton models for trees.

References

1 M. Bojańczyk. A bounding quantifier. In *CSL*, pages 41–55, 2004.
2 M. Bojańczyk. Weak MSO with the unbounding quantifier. In *STACS*, pages 159–170, 2009.
3 M. Bojańczyk. Beyond ω-regular languages. In *STACS*, pages 11–16, 2010.
4 M. Bojańczyk and T. Colcombet. Bounds in ω-regularity. In *LICS*, pages 285–296, 2006.
5 M. Bojańczyk and S. Toruńczyk. Deterministic automata and extensions of weak mso. In *FSTTCS*, pages 73–84, 2009.
6 M. Bojańczyk and S. Toruńczyk. Weak MSO+U over infinite trees. `http://www.mimuw.edu.pl/~bojan/papers/wmsou-trees.pdf`
7 J. Cabessa, J. Duparc, A. Facchini, and F. Murlak. The wadge hierarchy of max-regular languages. In *FSTTCS*, pages 121–132, 2009.
8 T. Colcombet. Factorisation forests for infinite words. In *FCT'07*, 2007.
9 T. Colcombet. The theory of stabilisation monoids and regular cost functions. In *ICALP (2)*, pages 139–150, 2009.
10 T. Colcombet, D. Kuperberg, and S. Lombardy. Regular temporal cost functions. In *ICALP (2)*, pages 563–574, 2010.
11 T. Colcombet and C. Löding. Regular cost functions over finite trees. In *LICS, 2010*
12 S. Hummel, M. Skrzypczak, and S. Toruńczyk. On the topological complexity of MSO+U and related automata models. In *MFCS*, pages 429–440, 2010.
13 J-E. Pin. Profinite methods in automata theory. In *STACS*, pages 31–50, 2009.
14 S. Toruńczyk. *Languages of profinite words and the limitedness problem.* PhD thesis, Warsaw University, 2011.