Alexander Kulikov
Nikolay Vereshchagin (Eds.)

LNCS 6651

Computer Science – Theory and Applications

6th International Computer Science Symposium in Russia, CSR 2011
St. Petersburg, Russia, June 2011, Proceedings

Springer

Alexander Kulikov   Nikolay Vereshchagin (Eds.)

Computer Science – Theory and Applications

6th International Computer Science Symposium in Russia, CSR 2011
St. Petersburg, Russia, June 14-18, 2011
Proceedings

Springer

Volume Editors

Alexander Kulikov
Steklov Institute of Mathematics at St. Petersburg
27 Fontanka, 191023 St. Petersburg, Russia
E-mail: kulikov@logic.pdmi.ras.ru

Nikolay Vereshchagin
Moscow State University
Department of Mathematical Logic and Theory of Algorithms
Leninskie gory 1, 119991 Moscow, Russia
E-mail: nikolay.vereshchagin@gmail.com

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-20711-2 e-ISBN 978-3-642-20712-9
DOI 10.1007/978-3-642-20712-9
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2011926308

CR Subject Classification (1998): F.2, F.3, E.3, G.2, F.1, F.4

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The 6th International Computer Science Symposium in Russia (CSR 2011) was held during June 14–18, 2011 in St. Petersburg, Russia, hosted by St. Petersburg Academic University of the Russian Academy of Sciences. It was the sixth event in the series of regular international meetings following CSR 2006 in St. Petersburg, CSR 2007 in Ekaterinburg, CSR 2008 in Moscow, CSR 2009 in Novosibirsk, and CSR 2010 in Kazan.

The opening lecture was given by Dima Grigoriev and eight other invited plenary lectures were given by Manindra Agrawal, László Babai, Andrei Bulatov, Jarkko Kari, Alexander Shen, Amir Shpilka, Madhu Sudan, and Sergey Yekhanin.

This volume contains the accepted papers and abstracts of most invited talks. The scope of the proposed topics for the symposium was quite broad and covered many areas of theoretical computer science and its applications. We received 76 papers in total, and out of these the Program Committee selected 29 papers for presentation at the symposium and for publication in the proceedings.

As usual, Yandex provided the Best Paper Awards; the recipients of these awards were selected by the Program Committee:

- Best Paper Award:
  Scott Aaronson, "The Equivalence of Sampling and Searching"
- Best Student Paper Award:
  Daniil Musatov, "Improving the Space-Bounded Version of Muchnik's Conditional Complexity Theorem via "Naive" Derandomization"

The reviewing process was organized using the EasyChair conference system, created by Andrei Voronkov. We would like to acknowledge that this system helped greatly to improve the efficiency of the committee work.

The following satellite events were co-located with CSR 2011:

- Workshop on Post-Quantum Cryptography
- Second Workshop on Program Semantics, Specification and Verification: Theory and Applications (PSSV 2011)
- Workshop on Universal Algebra and Computer Science (WUACS)

We are grateful to our sponsors:

- EMC Corporation
- Microsoft Research
- Russian Foundation for Basic Research
- Yandex (the largest Russian Internet portal providing key Web services)

We also thank the local organizers, in particular, Alexander V. Smal, Tatiana Vinogradova, and Nadya Zalesskaya.

June 2011

Alexander Kulikov
Nikolay Vereshchagin

Organization

CSR 2011 was organized by the Steklov Institute of Mathematics at St. Petersburg of the Russian Academy of Sciences and St. Petersburg Academic University of the Russian Academy of Sciences.

Program Committee Chair

Nikolay Vereshchagin	Moscow State University, Russia

Program Committee

Farid Ablayev	Kazan State University, Russia
Maxim Babenko	Moscow State University, Russia
Olivier Carton	Université Paris Diderot, France
Bruno Durand	Université de Provence, France
Anna Frid	Sobolev Institute of Mathematics, Russia
Valentine Kabanets	Simon Fraser University, Canada
Juhani Karhumäki	University of Turku, Finland
Michal Koucký	Institute of Mathematics, Czech Republic
Meena Mahajan	Institute of Mathematical Sciences, India
Yuri Matiyasevich	Steklov Institute of Mathematics at St. Petersburg, Russia
Pierre McKenzie	Université de Montréal, Canada
Ilya Mironov	Microsoft Research, USA
Ilan Newman	Haifa University, Israel
Alexander Razborov	University of Chicago, USA and Steklov Mathematical Institute, Russia
Miklos Santha	Université Paris-Sud, France
Nitin Saxena	Hausdorff Center for Mathematics, Germany
Valentin Shehtman	Institute for the Information Transmission Problems, Russia
Alexander Sherstov	Microsoft Research, USA
Thomas Thierauf	Aalen University, Germany
Oleg Verbitsky	Institute for Applied Problems of Mechanics and Mathematics, Ukraine
Mikhail Volkov	Ural State University, Russia
Igor Walukiewicz	Université de Bordeaux, France

Symposium Chair

Alexander Kulikov	Steklov Institute of Mathematics at St. Petersburg, Russia

CSR Steering Committee

Volker Diekert	University of Stuttgart, Germany
Anna Frid	Sobolev Institute of Mathematics, Russia
Edward A. Hirsch	Steklov Institute of Mathematics at St. Petersburg, Russia
Juhani Karhumäki	University of Turku, Finland
Mikhail Volkov	Ural State University, Russia

External Reviewers

Mikhail Abramskiy
Anil Ada
Luis Antunes
Sergei Avgustinovich
Matthias Baaz
Malte Beecken
Eli Ben-Sasson
Oren Ben-Zwi
Eugene Beschastnov
Laurent Bienvenu
Michael Blondin
Hans L. Bodlaender
Benedikt Bollig
Gerth Stølting Brodal
Peter Bro Miltersen
Harry Buhrman
Andrei Bulatov
Michaël Cadilhac
Krishnendu Chatterjee
Bruno Courcelle
Samir Datta
Holger Dell
Emilio Di Giacomo
Christoph Dürr
Henning Fernau
Jiri Fiala
Fedor Fomin
Martin Fürer
Naveen Garg
Hugo Gimbert
Nikolai Gravin
Alexey Gusakov
Vesa Halava
Magnus M. Halldorsson
Tero Harju
Johan Håstad
Danny Hermelin
Mika Hirvensalo
Emmanuel Jeandel
Artur Jez
Jarkko Kari
Marek Karpinski
Dmitry Karpov
Ralf Klasing
Ignat Kolesnichenko
Ilya Kornakov
Richard Kralovic
Andreas Krebs
Thorsten Kräling
Dietrich Kuske
Troy Lee
Jerome Leroux
Xiaowen Liu
Vadim Lozin
Johann Makowsky
Catherine Matias
Jochen Messner
George Metcalfe
Johannes Mittmann
Jerome Monnot
Pavel Nalivaiko
N.S. Narayanaswamy
Alexander Okhotin
Krzysztof Onak
Giorgio Orsi
Alexei Pastor
Wojciech Plandowski
Vladimir Podolskii
Jean-Yves Potvin
Ivan Pouzyrevsky
Pavel Pudlak
Mathieu Raffinot
Ilya Razenshteyn
Gaétan Richard
Adi Rosen
Peter Rossmanith
Andrei Rumyantsev
Wojciech Rytter
Aleksi Saarela
Pavel Salimov
Kai Salomaa
Olivier Serre
Rocco Servedio
Sergey Sevastyanov
Ilya Shapirovsky
Arseny Shur
Tatiana Starikovskaya
Tamon Stephen
Christopher Umans
Alexander Vasiliev
Fabian Wagner
Oren Weimann
Steve Wismath
Tatiana Yavorskaya

Table of Contents

The Equivalence of Sampling and Searching

Scott Aaronson[⋆]

MIT, Cambridge, MA, USA
aaronson@csail.mit.edu

Abstract. In a *sampling problem*, we are given an input $x \in \{0,1\}^n$, and asked to sample approximately from a probability distribution $\mathcal{D}_x$ over poly(n)-bit strings. In a *search problem*, we are given an input $x \in \{0,1\}^n$, and asked to find a member of a nonempty set A_x with high probability. (An example is finding a Nash equilibrium.) In this paper, we use tools from Kolmogorov complexity to show that sampling and search problems are "essentially equivalent." More precisely, for any sampling problem S, there exists a search problem R_S such that, if $\mathcal{C}$ is any "reasonable" complexity class, then R_S is in the search version of $\mathcal{C}$ if and only if S is in the sampling version. What makes this nontrivial is that the same R_S works for every $\mathcal{C}$.

As an application, we prove the surprising result that $\mathsf{SampP} = \mathsf{SampBQP}$ *if and only if* $\mathsf{FBPP} = \mathsf{FBQP}$. In other words, classical computers can efficiently sample the output distribution of every quantum circuit, if and only if they can efficiently solve every search problem that quantum computers can solve.

1 Introduction

The *Extended Church-Turing Thesis (ECT)* says that all computational problems that are feasibly solvable in the physical world are feasibly solvable by a probabilistic Turing machine. By now, there have been almost two decades of discussion about this thesis, and the challenge that quantum computing poses to it. This paper is about a related question that has attracted surprisingly little interest: namely, what exactly should we understand the ECT to *state*? When we say "all computational problems," do we mean decision problems? promise problems? search problems? sampling problems? possibly other types of problems? Could the ECT hold for some of these types of problems but fail for others?

Our main result is an *equivalence* between sampling and search problems: the ECT holds for one type of problem if and only if it holds for the other. As a motivating example, we will prove the surprising fact that, if classical

[⋆] I thank Alex Arkhipov for helpful discussions that motivated this work, Dana Moshkovitz for pointing me to Proposition 2 from [6], and the anonymous reviewers for their comments. This material is based upon work supported by the National Science Foundation under Grant No. 0844626. Also supported by a DARPA YFA grant, a TIBCO Chair, and the Sloan Foundation.

computers can efficiently solve any search problem that quantum computers can solve, then they can *also* approximately sample the output distribution of any quantum circuit. The proof makes essential use of Kolmogorov complexity. The technical tools that we will use are standard ones in the algorithmic information theory literature; our contribution is simply to apply those tools to obtain a useful equivalence principle in complexity theory that seems not to have been known before.

While the *motivation* for our equivalence theorem came from quantum computing, we wish to stress that the theorem itself is much more general, and has nothing to do with quantum computing in particular.

1.1 Background

Theoretical computer science has traditionally focused on *language decision problems*, where given a language $L \subseteq \{0,1\}^*$, the goal is to decide whether $x \in L$ for any input x. From this perspective, asking whether quantum computing contradicts the ECT is tantamount to asking:

Problem 1. Does $\mathsf{BPP} = \mathsf{BQP}$?

However, one can also consider *promise problems*, where the goal is to accept all inputs in a set $L_{\text{YES}} \subseteq \{0,1\}^*$ and reject all inputs in another set $L_{\text{NO}} \subseteq \{0,1\}^*$. Here L_{YES} and L_{NO} are disjoint, but their union is not necessarily all strings, and an algorithm can do whatever it likes on inputs not in $L_{\text{YES}} \cup L_{\text{NO}}$. Goldreich [4] has made a strong case that promise problems are at least as fundamental as language decision problems, if not more so. To give one relevant example, the task

Given a quantum circuit C, estimate the probability $p(C)$ that C accepts

is easy to formulate as a promise problem, but has no known formulation as a language decision problem. The reason is the usual "knife-edge" issue: given any probability $p^* \in [0,1]$ and error bound $\varepsilon \geq 1/\operatorname{poly}(n)$, we can ask a simulation algorithm to accept all quantum circuits C such that $p(C) \geq p^* + \varepsilon$, and to reject all circuits C such that $p(C) \leq p^* - \varepsilon$. But we cannot reasonably ask an algorithm to decide whether $p(C) = p^* + 2^{-n}$ or $p(C) = p^* - 2^{-n}$: if $p(C)$ is too close to p^*, then the algorithm's behavior is unknown.

Let $\mathsf{PromiseBPP}$ and $\mathsf{PromiseBQP}$ be the classes of promise problems solvable by probabilistic and quantum computers respectively, in polynomial time and with bounded probability of error. Then a second way to ask whether quantum mechanics contradicts the ECT is to ask:

Problem 2. Does $\mathsf{PromiseBPP} = \mathsf{PromiseBQP}$?

Now, if one accepts replacing languages by promise problems, then there seems little reason not to go further. One can also consider *search problems*, where given an input $x \in \{0,1\}^n$, the goal is to output any element of some nonempty "solution set" $A_x \subseteq \{0,1\}^{\text{poly}(n)}$.[1] Perhaps the most famous example of a search problem is *finding a Nash equilibrium*, which Daskalakis et al. [2] showed to be complete for the class PPAD. By Nash's Theorem, every game has at least one Nash equilibrium, but the problem of finding one has no known formulation as either a language decision problem or a promise problem.

Let FBPP and FBQP be the classes of search problems solvable by probabilistic and quantum computers respectively, with success probability $1-\delta$, in time polynomial in n and $1/\delta$.[2] Then a third version of the "ECT question" is:

Problem 3. Does FBPP = FBQP?

There is yet another important type of problem in theoretical computer science. These are *sampling problems*, where given an input $x \in \{0,1\}^n$, the goal is to sample (exactly or, more often, approximately) from some probability distribution $\mathcal{D}_x$ over poly (n)-bit strings. Well-known examples of sampling problems include sampling a random point in a high-dimensional convex body and sampling a random matching in a bipartite graph.

Let SampP and SampBQP be the classes of sampling problems that are solvable by probabilistic and quantum computers respectively, to within ε error in total variation distance, in time polynomial in n and $1/\varepsilon$.[3] Then a fourth version of our question is:

Problem 4. Does SampP = SampBQP?

Not surprisingly, *all* of the above questions are open. But we can ask an obvious meta-question:

> *What is the relationship among Problems 1-4? If the ECT fails in one sense, must it fail in the other senses as well?*

In one direction, there are some easy implications:

$$\begin{aligned}\mathsf{SampP} = \mathsf{SampBQP} &\Longrightarrow \mathsf{FBPP} = \mathsf{FBQP}\\ &\Longrightarrow \mathsf{PromiseBPP} = \mathsf{PromiseBQP}\\ &\Longrightarrow \mathsf{BPP} = \mathsf{BQP}.\end{aligned}$$

[1] Search problems are also called "relational problems," for the historical reason that one can define such a problem using a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, with $(x,y) \in R$ if and only if $y \in A_x$. Another name often used is "function problems." But that is inaccurate, since the desired output is *not* a function of the input, except in the special case $|A_x| = 1$. We find "search problems" to be the clearest name, and will use it throughout. The one important point to remember is that a search problem need *not* be an NP search problem: that is, solutions need not be efficiently verifiable.

[2] The F in FBPP and FBQP stands for "function problem." Here we are following the standard naming convention, even though the term "function problem" is misleading for the reason pointed out earlier.

[3] Note that we write SampP instead of "SampBPP" because there is no chance of confusion here. Unlike with decision, promise, and relation problems, with sampling problems it does not even make sense to talk about deterministic algorithms.

For the first implication, if every quantumly samplable distribution were also classically samplable, then given a quantum algorithm Q solving a search problem R, we could approximate Q's output distribution using a classical computer, and thereby solve R classically as well. For the second and third implications, every promise problem is also a search problem (with solution set $A_x \subseteq \{0,1\}$), and every language decision problem is also a promise problem (with the empty promise).

So the interesting part concerns the possible implications in the "other" direction. For example, could it be the case that $\mathsf{BPP} = \mathsf{BQP}$, yet $\mathsf{PromiseBPP} \neq \mathsf{PromiseBQP}$? Not only is this a formal possibility, but it does not even seem absurd, when we consider that

(1) the existing candidates for languages in $\mathsf{BQP} \setminus \mathsf{BPP}$ (for example, decision versions of the factoring and discrete log problems [7]) are all extremely "special" in nature, but
(2) $\mathsf{PromiseBQP}$ contains the "general" problem of estimating the acceptance probability of an arbitrary quantum circuit.

A second example of a difficult and unsolved meta-question is whether $\mathsf{PromiseBPP} = \mathsf{PromiseBQP}$ implies $\mathsf{SampP} = \mathsf{SampBQP}$. Translated into "physics language," the question is this: suppose we had an efficient classical algorithm to estimate the *expectation value* of any observable in quantum mechanics. Would that imply an efficient classical algorithm to *simulate any quantum experiment*, in the sense of sampling from a probability distribution close to the one quantum mechanics predicts?

1.2 Our Results

This paper shows that *two* of the four types of problem discussed above—namely, sampling problems and search problems—are "equivalent" in a very non-obvious sense. Specifically, given any sampling problem S, we will construct a search problem $R = R_S$ such that, if $\mathcal{C}$ is any "reasonable" model of computation, then S is in $\mathsf{Samp}\mathcal{C}$ (the sampling version of $\mathcal{C}$) if and only if R is in $\mathsf{F}\mathcal{C}$ (the search version of $\mathcal{C}$). Here is a more formal statement of the result:

Theorem 1 (Sampling/Searching Equivalence Theorem). *Let S be any sampling problem. Then there exists a search problem R_S such that*

(i) If $\mathcal{O}$ is any oracle for S, then $R_S \in \mathsf{FBPP}^{\mathcal{O}}$.
(ii) If B is any probabilistic Turing machine solving R_S, then $S \in \mathsf{SampP}^{B}$.

(Importantly, the same search problem R_S works for all $\mathcal{O}$ and B.)

As one application, we show that the "obvious" implication $\mathsf{SampP} = \mathsf{SampBQP} \Longrightarrow \mathsf{FBPP} = \mathsf{FBQP}$ can be reversed:

Theorem 2. $\mathsf{FBPP} = \mathsf{FBQP}$ *if and only if* $\mathsf{SampP} = \mathsf{SampBQP}$. *In other words, classical computers can efficiently solve every* FBQP *search problem, if and only if they can approximately sample the output distribution of every quantum circuit.*

As a second application (which was actually the original motivation for this work), we extend a recent result of Aaronson and Arkhipov [1]. These authors gave a sampling problem that is solvable using a simple linear-optics experiment (so in particular, in $\mathsf{SampBQP}$), but is *not* solvable efficiently by a classical computer under a plausible complexity assumption. More formally, consider the following problem, called $|\mathrm{GPE}|^2$ (the GPE stands for Gaussian Permanent Estimation):

Problem 5 ($|\mathrm{GPE}|^2$). Given a matrix $X \in \mathbb{C}^{n\times n}$ of independent $\mathcal{N}(0,1)$ Gaussians, output a real number y such that $\left| y - |\mathrm{Per}(X)|^2 \right| \leq \varepsilon \cdot n!$, with probability at least $1-\delta$ over $X \sim \mathcal{N}(0,1)_{\mathbb{C}}^{n\times n}$, in $\mathrm{poly}(n, 1/\varepsilon, 1/\delta)$ time.

The main result of [1] is the following:

Theorem 3 ([1]). $\mathsf{SampP} = \mathsf{SampBQP}$ *implies* $|\mathrm{GPE}|^2 \in \mathsf{FBPP}^{\mathsf{NP}}$.

The central conjecture made in [1] is that $|\mathrm{GPE}|^2$ is $\#\mathsf{P}$-complete. If this is the case, then $\mathsf{SampP} = \mathsf{SampBQP}$ would imply $\mathsf{P}^{\#\mathsf{P}} = \mathsf{BPP}^{\mathsf{NP}}$, which in turn would imply $\mathsf{PH} = \mathsf{BPP}^{\mathsf{NP}}$ by Toda's Theorem. Or to put it differently: *we could rule out a polynomial-time classical algorithm to sample the output distribution of a quantum computer, under the sole assumption that the polynomial hierarchy is infinite.*

Now, by using Theorem 2 from this paper, we can deduce, in a completely "automatic" way, that the counterpart of Theorem 3 must hold with *search* problems in place of sampling problems:

Corollary 1. $\mathsf{FBPP} = \mathsf{FBQP}$ *implies* $|\mathrm{GPE}|^2 \in \mathsf{FBPP}^{\mathsf{NP}}$. *So in particular, if* $|\mathrm{GPE}|^2$ *is* $\#\mathsf{P}$*-complete and* PH *is infinite, then* $\mathsf{FBPP} \neq \mathsf{FBQP}$.

1.3 Proof Overview

Let us explain the basic difficulty we need to overcome to prove Theorem 1. Given a probability distribution $\mathcal{D}_x$ over $\{0,1\}^{\mathrm{poly}(n)}$, we want to define a set $A_x \subseteq \{0,1\}^{\mathrm{poly}(n)}$, such that the ability to *find* an element of A_x is equivalent to the ability to *sample* from $\mathcal{D}_x$. At first glance, such a general reduction seems impossible. For let $R = \{A_x\}_x$ be the search problem in which the goal is to find an element of A_x given x. Then consider an oracle $\mathcal{O}$ that, on input x, returns the lexicographically first element of A_x. Such an oracle $\mathcal{O}$ certainly solves R, but it seems useless if our goal is to *sample* uniformly from the set A_x (or indeed, from any other interesting distribution related to A_x).

Our solution will require going outside the black-box reduction paradigm. In other words, given a sampling problem $S = \{\mathcal{D}_x\}_x$, we do *not* show that $S \in \mathsf{SampP}^{\mathcal{O}}$, where $\mathcal{O}$ is any oracle that solves the corresponding search problem R_S. Instead, we use the fact that $\mathcal{O}$ is computed by a Turing machine. We then define R_S in such a way that $\mathcal{O}$ must return, not just any element in the support of $\mathcal{D}_x$, but an element with *near-maximal Kolmogorov complexity.*

The idea here is simple: if a Turing machine B is probabilistic, then it can certainly output a string y with high Kolmogorov complexity, by just generating y at random. But the converse also holds: if B outputs a string y with high Kolmogorov complexity, then y *must* have been generated randomly. For otherwise, the code of B would constitute a succinct description of y.

Given any set $A \subseteq \{0,1\}^n$, it is not hard to use the above "Kolmogorov trick" to force a probabilistic Turing machine B to sample almost-uniformly from A. We simply ask B to produce k samples $y_1, \ldots, y_k \in A$, for some $k = \text{poly}(n)$, such that the tuple $\langle y_1, \ldots, y_k \rangle$ has Kolmogorov complexity close to $k \log_2 |A|$. Then we output y_i for a uniformly random $i \in [k]$.

However, one can also generalize the idea, to force B to sample from an *arbitrary* distribution $\mathcal{D}$, not necessarily uniform. One way of doing this would be to reduce to the uniform case, by dividing the support of $\mathcal{D}$ into $\text{poly}(n)$ "buckets," such that $\mathcal{D}$ is nearly-uniform within each bucket, and then asking B to output Kolmogorov-random elements in each bucket. In this paper, however, we will follow a more direct approach, which exploits the beautiful known connection between Kolmogorov complexity and Shannon information. In particular, we will use the notion of a *universal randomness test* from algorithmic information theory [5,3]. Let $\mathcal{U}$ be the "universal prior," in which each string $y \in \{0,1\}^*$ occurs with probability proportional to $2^{-K(y)}$, where $K(y)$ is the prefix-free Kolmogorov complexity of y. Then given any computable distribution $\mathcal{D}$ and fixed string y, the universal randomness test provides a way to decide whether y was "plausibly drawn from $\mathcal{D}$," by considering the ratio $\Pr_{\mathcal{D}}[y] / \Pr_{\mathcal{U}}[y]$. The main technical fact we need to prove is simply that such a test can be applied in our *complexity-theoretic* context, where we care (for example) that the number of samples from $\mathcal{D}$ scales polynomially with the inverses of the relevant error parameters.

From one perspective, our result represents a surprising use of Kolmogorov complexity in the seemingly "distant" realm of polynomial-time reductions. Let us stress that we are *not* using Kolmogorov complexity as just a technical convenience, or as shorthand for a counting argument. Rather, Kolmogorov complexity seems essential even to define a search problem R_S with the properties we need. From another perspective, however, our use of Kolmogorov complexity is close in spirit to the reasons why Kolmogorov complexity was defined and studied in the first place! The whole point, after all, is to be able to talk about the "randomness of an individual object," without reference to any distribution from which the object was drawn. And that is exactly what we need, if we want to achieve the "paradoxical" goal of sampling from a distribution, using an oracle that is guaranteed only to output a *fixed* string y with specified properties.

2 Preliminaries

2.1 Sampling and Search Problems

We first formally define sampling problems, as well as the complexity classes SampP and $\mathsf{SampBQP}$.

Definition 1 (Sampling Problems, SampP, and SampBQP). *A sampling problem S is a collection of probability distributions $(\mathcal{D}_x)_{x\in\{0,1\}^*}$, one for each input string $x \in \{0,1\}^n$, where $\mathcal{D}_x$ is a distribution over $\{0,1\}^{p(n)}$, for some fixed polynomial p. Then* SampP *is the class of sampling problems $S = (\mathcal{D}_x)_{x\in\{0,1\}^*}$ for which there exists a probabilistic polynomial-time algorithm B that, given $\langle x, 0^{1/\varepsilon}\rangle$ as input, samples from a probability distribution $\mathcal{C}_x$ such that $\|\mathcal{C}_x - \mathcal{D}_x\| \leq \varepsilon$.* SampBQP *is defined the same way, except that B is a quantum algorithm rather than a classical one.*

Let us also define search problems, as well as the complexity classes FBPP and FBQP.

Definition 2 (Search Problems, FBPP, and FBQP). *A search problem R is a collection of nonempty sets $(A_x)_{x\in\{0,1\}^*}$, one for each input string $x \in \{0,1\}^n$, where $A_x \subseteq \{0,1\}^{p(n)}$ for some fixed polynomial p. Then* FBPP *is the class of search problems $R = (A_x)_{x\in\{0,1\}^*}$ for which there exists a probabilistic polynomial-time algorithm B that, given an input $x \in \{0,1\}^n$ together with $0^{1/\varepsilon}$, produces an output y such that*

$$\Pr[y \in A_x] \geq 1 - \varepsilon,$$

where the probability is over B's internal randomness. FBQP *is defined the same way, except that B is a quantum algorithm rather than a classical one.*

2.2 Algorithmic Information Theory

We now review some basic definitions and results from the theory of Kolmogorov complexity. Recall that a set of strings $P \subset \{0,1\}^*$ is called *prefix-free* if no $x \in P$ is a prefix of any other $y \in P$.

Definition 3 (Kolmogorov complexity). *Fix a universal Turing machine U, such that the set of valid programs of U is prefix-free. Then $K(y)$, or the prefix-free Kolmogorov complexity of y, is the minimum length of a program x such that $U(x) = y$. We can also define the conditional Kolmogorov complexity $K(y|z)$, as the minimum length of a program x such that $U(\langle x, z\rangle) = y$.*

We are going to need two basic lemmas that relate Kolmogorov complexity to standard information theory, and that can be found in the book of Li and Vitányi [5] for example. The first lemma follows almost immediately from Shannon's noiseless channel coding theorem.

Lemma 1. *Let $\mathcal{D} = \{p_x\}$ be any computable distribution over strings, and let x be any element in the support of $\mathcal{D}$. Then*

$$K(x) \leq \log_2 \frac{1}{p_x} + K(\mathcal{D}) + O(1),$$

where $K(\mathcal{D})$ represents the length of the shortest program to sample from $\mathcal{D}$. The same holds if we replace $K(x)$ and $K(\mathcal{D})$ by $K(x|y)$ and $K(\mathcal{D}|y)$ respectively, for any fixed y.

The next lemma follows from a counting argument.

Lemma 2 ([5]). *Let $\mathcal{D} = \{p_x\}$ be any distribution over strings (not necessarily computable). Then there exists a universal constant b such that*

$$\Pr_{x \sim \mathcal{D}} \left[K(x) \geq \log_2 \frac{1}{p_x} - c \right] \geq 1 - \frac{b}{2^c}.$$

The same holds if we replace $K(x)$ by $K(x|y)$ for any fixed y.

2.3 Information Theory

This section reviews some basic definitions and facts from information theory. Let $\mathcal{A} = \{p_x\}_x$ and $\mathcal{B} = \{q_x\}_x$ be two probability distributions over $[N]$. Then recall that the *variation distance* between $\mathcal{A}$ and $\mathcal{B}$ is defined as

$$\|\mathcal{A} - \mathcal{B}\| := \frac{1}{2} \sum_{x=1}^{N} |p_x - q_x|,$$

while the *KL-divergence* is

$$D_{KL}(\mathcal{A}||\mathcal{B}) := \sum_{x=1}^{N} p_x \log_2 \frac{p_x}{q_x}.$$

The variation distance and the KL-divergence are related as follows:

Proposition 1 (Pinsker's Inequality). $\|\mathcal{A} - \mathcal{B}\| \leq \sqrt{2 D_{KL}(\mathcal{A}||\mathcal{B})}$.

We will also need a fact about KL-divergence that has been useful in the study of parallel repetition, and that can be found (for example) in a paper by Rao [6].

Proposition 2 ([6]). *Let $\mathcal{R}$ be a distribution over $[N]^k$, with marginal distribution $\mathcal{R}_i$ on the i^{th} coordinate. Let $\mathcal{D}$ be a distribution over $[N]$. Then*

$$\sum_{i=1}^{k} D_{KL}(\mathcal{R}_i||\mathcal{D}) \leq D_{KL}\left(\mathcal{R}||\mathcal{D}^k\right)$$

3 Main Result

Let $S = \{\mathcal{D}_x\}_x$ be a sampling problem. Then our goal is to construct a search problem $R = R_S = \{A_x\}_x$ that is "equivalent" to S. Given an input of the form $\langle x, 0^{1/\delta} \rangle$, the goal in the search problem will be to produce an output Y such that $Y \in A_{x,\delta}$, with success probability at least $1 - \delta$. The running time should be $\operatorname{poly}(n, 1/\delta)$.

Fix an input $x \in \{0,1\}^n$, and let $\mathcal{D} := \mathcal{D}_x$ be the corresponding probability distribution over $\{0,1\}^m$. Let $p_y := \Pr_{\mathcal{D}}[y]$ be the probability of y. We now define the search problem R. Let $N := m/\delta^{2.1}$, and let $Y = \langle y_1, \ldots, y_N \rangle$ be an N-tuple of m-bit strings. Then we set $Y \in A_{x,\delta}$ if and only if

$$\log_2 \frac{1}{p_{y_1} \cdots p_{y_N}} \leq K(Y \mid x, \delta) + \beta,$$

where $\beta := 1 + \log_2 1/\delta$.

The first thing we need to show is that any algorithm that solves the sampling problem S also solves the search problem R with high probability.

Lemma 3. *Let $\mathcal{C} = \mathcal{C}_x$ be any distribution over $\{0,1\}^m$ such that $\|\mathcal{C} - \mathcal{D}\| \leq \varepsilon$. Then*

$$\Pr_{Y \sim \mathcal{C}^N}[Y \notin A_{x,\delta}] \leq \varepsilon N + \frac{b}{2^\beta}.$$

Proof. We have

$$\Pr_{Y \sim \mathcal{C}^N}[Y \notin A_{x,\delta}] \leq \Pr_{Y \sim \mathcal{D}^N}[Y \notin A_{x,\delta}] + \left\|\mathcal{C}^N - \mathcal{D}^N\right\| \leq \Pr_{Y \sim \mathcal{D}^N}[Y \notin A_{x,\delta}] + \varepsilon N.$$

So it suffices to consider a Y drawn from $\mathcal{D}^N$. By Lemma 2,

$$\Pr_{Y \sim \mathcal{D}^N}\left[K(Y \mid x, \delta) \geq \log_2 \frac{1}{p_{y_1} \cdots p_{y_N}} - \beta\right] \geq 1 - \frac{b}{2^\beta}$$

Therefore

$$\Pr_{Y \sim \mathcal{D}^N}[Y \notin A_{x,\delta}] \leq \frac{b}{2^\beta},$$

and we are done.

The second thing we need to show is that any algorithm that solves the search problem R also samples from a distribution that is close to $\mathcal{D}$ in variation distance.

Lemma 4. *Let B be a probabilistic Turing machine, which given input $\langle x, 0^{1/\delta} \rangle$ outputs an N-tuple $Y = \langle y_1, \ldots, y_N \rangle$ of m-bit strings. Suppose that*

$$\Pr\left[B\left(x, 0^{1/\delta}\right) \in A_{x,\delta}\right] \geq 1 - \delta,$$

where the probability is over B's internal randomness. Let $\mathcal{R} = \mathcal{R}_x$ be the distribution over outputs of $B(x)$, and let $\mathcal{C} = \mathcal{C}_x$ be the distribution over $\{0,1\}^m$ that is obtained by from $\mathcal{R}$ by choosing one of the y_i's uniformly at random. Then there exists a constant Q_B, depending on B, such that

$$\|\mathcal{C} - \mathcal{D}\| \leq \delta + Q_B \sqrt{\frac{\beta}{N}}.$$

Proof. Let $\mathcal{R}'$ be a distribution that is identical to $\mathcal{R}$, except that we condition on $B\left(x, 0^{1/\delta}\right) \in A_{x,\delta}$. Then by hypothesis, $\|\mathcal{R} - \mathcal{R}'\| \leq \delta$. Now let $\mathcal{R}'_i$ be the marginal distribution of $\mathcal{R}'$ on the i^{th} coordinate, and let

$$\mathcal{C}' = \frac{1}{N}\sum_{i=1}^{N}\mathcal{R}'_i$$

be the distribution over $\{0,1\}^m$ that is obtained from $\mathcal{R}'$ by choosing one of the y_i's uniformly at random. Then clearly $\|\mathcal{C}-\mathcal{C}'\| \leq \delta$ as well. So by the triangle inequality,

$$\|\mathcal{C}-\mathcal{D}\| \leq \|\mathcal{C}-\mathcal{C}'\| + \|\mathcal{C}'-\mathcal{D}\| \leq \delta + \|\mathcal{C}'-\mathcal{D}\|,$$

and it suffices to upper-bound $\|\mathcal{C}'-\mathcal{D}\|$.

Let $q_Y := \Pr_{\mathcal{R}'}[Y]$. Then by Lemma 1,

$$K(Y \mid x,\delta) \leq \log_2 \frac{1}{q_Y} + K(\mathcal{R}') + O(1)$$

for all $Y \in (\{0,1\}^m)^N$. Also, since $Y \in A_{x,\delta}$, by assumption we have

$$\log_2 \frac{1}{p_{y_1}\cdots p_{y_N}} \leq K(Y \mid x,\delta) + \beta.$$

Combining,

$$\log_2 \frac{1}{p_{y_1}\cdots p_{y_N}} \leq \log_2 \frac{1}{q_Y} + K(\mathcal{R}') + O(1) + \beta.$$

This implies the following upper bound on the KL-divergence:

$$\begin{aligned} D_{KL}\left(\mathcal{R}' || \mathcal{D}^N\right) &= \sum_{Y\in(\{0,1\}^m)^N} q_Y \log_2 \frac{q_Y}{p_{y_1}\cdots p_{y_N}} \\ &\leq \max_Y \log_2 \frac{q_Y}{p_{y_1}\cdots p_{y_N}} \\ &\leq K(\mathcal{R}') + O(1) + \beta. \end{aligned}$$

So by Proposition 2,

$$\sum_{i=1}^{N} D_{KL}(\mathcal{R}'_i || \mathcal{D}) \leq D_{KL}\left(\mathcal{R}' || \mathcal{D}^N\right) \leq K(\mathcal{R}') + O(1) + \beta,$$

and by Proposition 1,

$$\frac{1}{2}\sum_{i=1}^{N} \|\mathcal{R}'_i - \mathcal{D}\|^2 \leq K(\mathcal{R}') + O(1) + \beta.$$

So by Cauchy-Schwarz,

$$\sum_{i=1}^{N} \|\mathcal{R}'_i - \mathcal{D}\| \leq \sqrt{N\left(2\beta + 2K(\mathcal{R}') + O(1)\right)}.$$

Hence

$$\|\mathcal{C}' - \mathcal{D}\| \leq \sqrt{\frac{2\beta + 2K(\mathcal{R}') + O(1)}{N}},$$

and

$$\|\mathcal{C} - \mathcal{D}\| \leq \|\mathcal{C} - \mathcal{C}'\| + \|\mathcal{C}' - \mathcal{D}\| \leq \delta + \sqrt{\frac{2\beta + 2K(\mathcal{R}') + O(1)}{N}} \leq \delta + Q_B\sqrt{\frac{\beta}{N}},$$

for some constant Q_B depending on B.

By combining Lemmas 3 and 4, we can now prove Theorem 1: that for any sampling problem $S = (\mathcal{D}_x)_{x\in\{0,1\}^*}$ (where $\mathcal{D}_x$ is a distribution over $m = m(n)$-bit strings), there exists a search problem $R_S = (A_x)_{x\in\{0,1\}^*}$ that is "equivalent" to S in the following two senses.

(i) Let $\mathcal{O}$ be any oracle that, given $\langle x, 0^{1/\varepsilon}, r\rangle$ as input, outputs a sample from a distribution $\mathcal{C}_x$ such that $\|\mathcal{C}_x - \mathcal{D}_x\| \leq \varepsilon$, as we vary the random string r. Then $R_S \in \mathsf{FBPP}^{\mathcal{O}}$.

(ii) Let B be any probabilistic Turing machine that, given $\langle x, 0^{1/\delta}\rangle$ as input, outputs a $Y \in (\{0,1\}^m)^N$ such that $Y \in A_{x,\delta}$ with probability at least $1-\delta$. Then $S \in \mathsf{SampP}^B$.

Proof (Proof of Theorem 1 (Sampling/Searching Equivalence Theorem)). For part (i), given an input $\langle x, 0^{1/\delta}\rangle$, suppose we want to output an N-tuple $Y = \langle y_1, \ldots, y_N\rangle \in (\{0,1\}^m)^N$ such that $Y \in A_{x,\delta}$, with success probability at least $1-\delta$. Recall that $N = m/\delta^{2.1}$. Then the algorithm is this:

(1) Set $\varepsilon := \frac{\delta}{2N} = \frac{\delta^{3.1}}{2m}$.

(2) Call $\mathcal{O}$ on inputs $\langle x, 0^{1/\varepsilon}, r_1\rangle, \ldots, \langle x, 0^{1/\varepsilon}, r_N\rangle$, where $r_1, \ldots, r_N$ are independent random strings, and output the result as $Y = \langle y_1, \ldots, y_N\rangle$.

Clearly this algorithm runs in poly $(n, 1/\delta)$ time. Furthermore, by Lemma 3, its failure probability is at most

$$\varepsilon N + \frac{b}{2^\beta} \leq \delta.$$

For part (ii), given an input $\langle x, 0^{1/\varepsilon}\rangle$, suppose we want to sample from a distribution $\mathcal{C}_x$ such that $\|\mathcal{C}_x - \mathcal{D}_x\| \leq \varepsilon$. Then the algorithm is this:

(1) Set $\delta := \varepsilon/2$, so that $N = m/\delta^{2.1} = \Theta\left(m/\varepsilon^{2.1}\right)$.

(2) Call B on input $\langle x, 0^{1/\delta}\rangle$, and let $Y = \langle y_1, \ldots, y_N\rangle$ be B's output.

(3) Choose $i \in [N]$ uniformly at random, and output y_i as the sample from $\mathcal{C}_x$.

Clearly this algorithm runs in poly $(n, 1/\varepsilon)$ time. Furthermore, by Lemma 4 we have

$$\|\mathcal{C}_x - \mathcal{D}_x\| \leq \delta + Q_B\sqrt{\frac{\beta}{N}} \leq \frac{\varepsilon}{2} + Q_B\sqrt{\frac{\varepsilon^{2.1}(2 + \log 1/\varepsilon)}{m}},$$

for some constant Q_B depending only on B. So in particular, there exists a constant C_B such that $\|\mathcal{C}_x - \mathcal{D}_x\| \leq \varepsilon$ for all $m \geq C_B$. For $m < C_B$, we can simply hardwire a description of $\mathcal{D}_x$ for every x into the algorithm (note that the algorithm can depend on B; we do not need a single algorithm that works for all B's simultaneously).

In particular, Theorem 1 means that $S \in \mathsf{SampP}$ if and only if $R_S \in \mathsf{FBPP}$, and likewise $S \in \mathsf{SampBQP}$ if and only if $R_S \in \mathsf{FBQP}$, and so on for any model of computation that is "below recursive" (i.e., simulable by a Turing machine) and has the extremely simple closure properties used in the proof.

3.1 Implication for Quantum Computing

We now apply Theorem 1 to prove Theorem 2, that $\mathsf{SampP} = \mathsf{SampBQP}$ if and only if $\mathsf{FBPP} = \mathsf{FBQP}$.

Proof (Proof of Theorem 2). First, suppose $\mathsf{SampP} = \mathsf{SampBQP}$. Then consider a search problem $R = (A_x)_x$ in FBQP. By assumption, there exists a polynomial-time quantum algorithm Q that, given $\langle x, 0^{1/\delta} \rangle$ as input, outputs a y such that $y \in A_x$ with probability at least $1 - \delta$. Let $\mathcal{D}_{x,\delta}$ be the probability distribution over y's output by Q on input $\langle x, 0^{1/\delta} \rangle$. Then to solve R in FBPP, clearly it suffices to sample approximately from $\mathcal{D}_{x,\delta}$ in classical polynomial time. But we can do this by the assumption that $\mathsf{SampP} = \mathsf{SampBQP}$.[4]

Second, suppose $\mathsf{FBPP} = \mathsf{FBQP}$. Then consider a sampling problem S in $\mathsf{SampBQP}$. By Theorem 1, we can define a search counterpart R_S of S, such that

$$S \in \mathsf{SampBQP} \Longrightarrow R_S \in \mathsf{FBQP} \Longrightarrow R_S \in \mathsf{FBPP} \Longrightarrow S \in \mathsf{SampP}.$$

Hence $\mathsf{SampP} = \mathsf{SampBQP}$.

Theorem 2 is easily seen to relativize: for all oracles A, we have $\mathsf{SampP}^A = \mathsf{SampBQP}^A$ if and only if $\mathsf{FBPP}^A = \mathsf{FBQP}^A$. (Of course, when proving a relativized version of Theorem 1, we have to be careful to define the search problem R_S using Kolmogorov complexity for Turing machines with A-oracles.)

4 Extensions and Open Problems

4.1 Equivalence of Sampling and *Decision* Problems?

Perhaps the most interesting question we leave open is whether any nontrivial equivalence holds between sampling (or search) problems on the one hand, and *decision* or *promise* problems on the other. One way to approach this question is as follows: does there exist a sampling problem S that is provably *not* equivalent to any decision problem, in the sense that for every language $L \subseteq \{0,1\}^*$, either $S \notin \mathsf{SampP}^L$, or else there exists an oracle $\mathcal{O}$ solving

[4] As mentioned in Section 1, the same argument shows that $\mathsf{SampP} = \mathsf{SampBQP}$ (or equivalently, $\mathsf{FBPP} = \mathsf{FBQP}$) implies $\mathsf{BPP} = \mathsf{BQP}$. However, the converse is far from clear: we have no idea whether $\mathsf{BPP} = \mathsf{BQP}$ implies $\mathsf{SampP} = \mathsf{SampBQP}$.

S such that $L \notin \mathsf{BPP}^{\mathcal{O}}$? What if we require the oracle $\mathcal{O}$ to be computable? As far as we know, these questions are open.

One might object that, given any sampling problem S, it is easy to define a language L_S that is "equivalent" to S, using standard Turing-machine enumeration tricks. Even more vacuously, one could ensure ensure $S \in \mathsf{SampP} \iff L_S \in \mathsf{P}$ in the following "tautological" way:

> *"Take L_S to be the empty language if $S \in \mathsf{SampP}$, or an EXP-complete language if $S \notin \mathsf{SampP}$!"*

In our view, the problem with both of these approaches is that they fail to *reduce* the sampling problem S to the language L_S or vice versa. Of course, Theorem 1 did not quite reduce S to the search problem R_S either. However, Theorem 1 came "close enough" to giving a reduction that we were able to use it to derive interesting consequences for complexity theory, such as $\mathsf{SampP} = \mathsf{SampBQP}$ if and only if $\mathsf{FBPP} = \mathsf{FBQP}$. If we attempted to prove similar consequences using a language L_S like the one above, then we would end up with a *different* language L_S, depending on whether our starting assumption was $S \in \mathsf{SampP}$, $S \in \mathsf{SampBQP}$, or some other assumption. By contrast, Theorem 1 constructed a *single* search problem R_S that is equivalent to S in the classical model, the quantum model, and every other "reasonable" computational model.

4.2 Was Kolmogorov Complexity Necessary?

Could we have proved Theorem 1 *without* using Kolmogorov complexity or anything like it? One way to formalize this question is as follows: does there exist a sampling problem S such that, for every search problem R, either there exists an oracle $\mathcal{O}$ solving S such that $R \notin \mathsf{FBPP}^{\mathcal{O}}$, or there exists an oracle $\mathcal{O}$ solving R such that $S \notin \mathsf{SampP}^{\mathcal{O}}$? Notice that, if R is the search problem from Theorem 1, then the latter oracle (if it exists) must be uncomputable. Thus, we are essentially asking whether the computability assumption in Theorem 1 was necessary.

4.3 From Search Problems to Sampling Problems

Theorem 1 showed how to take any sampling problem S, and define a search problem $R = R_S$ that is equivalent to S. Can one go the other direction? That is, given a search problem R, can one define a sampling problem $S = S_R$ that is equivalent to R? The following theorem is the best we were able to find in this direction.

Theorem 4. *Let $R = (A_x)_x$ be any search problem. Then there exists a sampling problem $S_R = \{\mathcal{D}_x\}_x$ that is "almost equivalent" to R, in the following senses.*

(i) If $\mathcal{O}$ is any oracle solving S_R, then $R \in \mathsf{FBPP}^{\mathcal{O}}$.
(ii) If B is any probabilistic Turing machine solving R, then there exists a constant $\eta_B > 0$ such that a SampP^B machine can sample from a probability distribution $\mathcal{C}_x$ with $\|\mathcal{C}_x - \mathcal{D}_x\| \leq 1 - \eta_B$.

We see it as an interesting problem whether Theorem 4 still holds with the condition $\|\mathcal{C}_x - \mathcal{D}_x\| \leq 1 - \eta_B$ replaced by $\|\mathcal{C}_x - \mathcal{D}_x\| \leq \varepsilon$ (in other words, with $S_R \in \mathsf{SampP}^B$).

4.4 Making the Search Problem Checkable

One obvious disadvantage of Theorem 1 is that the search problem $R = (A_x)_x$ is defined using Kolmogorov complexity, which is uncomputable. In particular, there is no algorithm to decide whether $y \in A_x$. However, it is not hard to fix this problem, by replacing the Kolmogorov complexity with the *time-bounded* or *space-bounded* Kolmogorov complexities in our definition of R. The price is that we then also have to assume a complexity bound on the Turing machine B in the statement of Theorem 1. In more detail:

Theorem 5. *Let S be any sampling problem, and let f be a time-constructible function. Then there exists a search problem $R_S = (A_x)_x$ such that*

(i) If $\mathcal{O}$ is any oracle solving S, then $R_S \in \mathsf{FBPP}^{\mathcal{O}}$.
(ii) If B is any $\mathsf{BPTIME}(f(n))$ Turing machine solving R_S, then $S \in \mathsf{SampP}^B$.
(iii) There exists a $\mathsf{SPACE}\left(f(n) + n^{O(1)}\right)$ algorithm to decide whether $y \in A_x$, given x and y.

In Theorem 5, how far can we decrease the computational complexity of R_S? It seems unlikely that one could check in NP (or $\mathsf{NTIME}\left(f(n) + n^{O(1)}\right)$) whether $y \in A_x$, for a search problem $R_S = \{A_x\}_x$ equivalent to S, but can we give formal evidence against this possibility?

References

1. Aaronson, S., Arkhipov, A.: The computational complexity of linear optics. In: Proc. ACM STOC, To appear. ECCC TR10-170, arXiv:1011.3245 (to appear, 2011)
2. Daskalakis, C., Goldberg, P.W., Papadimitriou, C.H.: The complexity of computing a Nash equilibrium. ACM Commun. 52(2), 89–97 (2009); Earlier version in Proceedings of STOC 2006
3. Gács, P.: Lecture notes on descriptional complexity and randomness (2010), `www.cs.bu.edu/~gacs/papers/ait-notes.pdf`
4. Goldreich, O.: On promise problems: a survey. In: Essays in Memory of Shimon Even, pp. 254–290, ECCC TR05-018 (2006)
5. Li, M., Vitányi, P.M.B.: An Introduction to Kolmogorov Complexity and Its Applications, 3rd edn. Springer, Heidelberg (2008)
6. Rao, A.: Parallel repetition in projection games and a concentration bound. In: Proc. ACM STOC, pp. 1–10, ECCC TR08-013 (2008)
7. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. 26(5), 1484–1509 (1997); Earlier version in IEEE FOCS 1994. quant-ph/9508027

Towards a Complexity Theory of Randomized Search Heuristics: Ranking-Based Black-Box Complexity

Benjamin Doerr and Carola Winzen[⋆]

Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany

Abstract. Randomized search heuristics are a broadly used class of general-purpose algorithms. Analyzing them via classical methods of theoretical computer science is a growing field. A big step forward would be a useful complexity theory for such algorithms. We enrich the two existing black-box complexity notions due to Wegener and other authors by the restrictions that not actual objective values, but only the relative quality of the previously evaluated solutions may be taken into account by the algorithm. Many randomized search heuristics belong to this class of algorithms. We show that the new ranking-based model gives more realistic complexity estimates for some problems, while for others the low complexities of the previous models still hold.

1 Introduction

Randomized search heuristics are general purpose algorithms to solve optimization problems. They include fancy approaches like evolutionary algorithms and ant colony optimization, but also classical approaches like random search or randomized hill-climbers.

In practice, randomized search heuristics often are surprisingly successful (and thus extensively used). They have the additional advantage that not too much understanding of the optimization problem at hand is needed, and that once implemented, they can easily be re-used for similar problems.

One of the difficulties in using such heuristics is that it is very hard to predict which problems are easy for a suitable heuristic and which are generally intractable for randomized search heuristics. There has been some theoretical work in the last 20 years, pioneered by the work of Ingo Wegener.

This work mostly lead to results for particular problems and particular heuristics, many of which disproved what was common believe of practitioner in this area. For example, Horn, Goldberg, and Deb [HGD94] showed that there are unimodal functions $f : \{0,1\}^n \to \mathbb{R}$ (that is, each search point $x \in \{0,1\}^n$ apart from the optimum has a Hamming neighbor with strictly better f-value) such that many commonly used search heuristics need time exponential in n to find the optimum of f.

[⋆] Carola Winzen is a recipient of the Google Europe Fellowship in Randomized Algorithms, and this work is supported in part by this Google Fellowship.

Still, for a broader understanding of what are easy and difficult problems, a complexity theory similar to what exists in classical algorithmics would be highly desirable, also for randomized search heuristics. The seminal paper by Droste, Jansen, and Wegener [DJW06], introducing the so-called black-box model, appears to be the first attempt to start such a complexity theory in the randomized search heuristics community. Earlier related work exists, e.g., [LTT89], [HK78], and [Ald83].

The paradigm that randomized search heuristics should ideally be problem-independent implies that the only way a randomized search heuristics can obtain problem-specific information is by evaluating a solution candidate. This evaluation is done by an oracle that returns the objective value, but reveals no further information on the objective function. An algorithms that has no access to the objective function (and thus the optimization problem to be solved) other than by querying the objective value from such an oracle, is called a *black-box algorithm*.

Given a class of functions $\mathcal{F}$, Droste et al. define the black-box complexity of $\mathcal{F}$ to be the the minimum (taken over all black-box algorithms) expected number of function evaluations needed to optimize any function $f \in \mathcal{F}$ (minimum worst-case runtime). This number, naturally, is a lower bound on the run time of any randomized search heuristics for the class $\mathcal{F}$, including evolutionary algorithms, ant colony approaches, simulated annealing, et cetera.

Unfortunately, it turned out that allowing all black-box algorithms leads to sometimes unexpectedly small complexities (obtained by not very sensible algorithms). As a trivial example, note that the black-box complexity of any class of functions $\mathcal{F} = \{f\}$ consisting just of a single objective function, is one — as certified by the algorithm that simply queries the optimum of f. This and further examples suggest that a restriction of the class of algorithms might lead to more meaningful results.

A major step towards this direction is the work by Lehre and Witt [LW10]. They introduce a so-called *unbiased black-box model*, which, among other restrictions, requires that all search points queried by the algorithms must be obtained from previous or random search points by so-called *unbiased variation operators*, see Section 2 for the full details. This in particular restricts an unwanted use of knowledge on the function class $\mathcal{F}$. It also leads to a lower bound of $\Omega(n \log n)$ for the complexity of all single-element classes $\mathcal{F} = \{f\}$, f having a unique global optimum, when only unary operators are allowed. This is, indeed, the typical run time of simple search heuristics like randomized hill-climbers on simple function classes like monotone functions.

In this work, we shall argue that the unbiased model of Lehre and Witt is still not restrictive enough. Let $f : \{0,1\}^n \to \mathbb{R}, x \mapsto \sum_{i=1}^{n} 2^{i-1} x_i$ be the binary-value function of the bit-string x. Let $\mathcal{F}$ be the the class of functions consisting of f and all functions obtained from f by permuting the order of the bit positions and by flipping the meaning of the values of some bit-positions. Then, as we shall show in this paper, the unbiased black-box complexity of $\mathcal{F}$ is only $\lceil \log_2 n \rceil + 2$. The corresponding algorithm (see Section 5) heavily exploits particular objective values which it learns from its queries.

This is what most randomized search heuristics do not do. They typically only use the objective values to compare search points. We define the black-box complexity notion referring to this paradigm by allowing the algorithms to only exploit the relative ranking of the search points queried so far. In other words, throughout the optimization the algorithms knows for any two already queried search points x and y no more than whether $f(x) < f(y)$, $f(x) = f(y)$ or $f(x) > f(y)$. In particular, it does not know the true values of $f(x)$ and $f(y)$. Still, this model captures many commonly used randomized search heuristics.

We show that our proposed model solves some drawbacks of the previous models. For example, for the binary-value function class introduced above, this ranking-based black-box complexity is of order $\Theta(n)$ instead of only $O(\log n)$ without the ranking restriction. In the $\Theta(n)$ statement, the lower bound proof is clearly the more interesting one. The upper bound is easily verified by a simple hill-climber that, in arbitrary order, changes a single bit-value of the current solution and accepts the new solution if it is better than the previous one. In summary, we see that for this function class, the ranking-based black-box complexity seems to give us a more useful complexity measure than the previous approaches.

We also analyze a second class commonly regarded in this context. Let $\mathcal{F}$ be the class of all (so-called onemax-) functions $f_z : \{0,1\}^n \to \mathbb{R}; x \mapsto n - \|x - z\|_1$, $z \in \{0,1\}$. Hence, $f_z(x)$ is just the number of bit positions x and z agree on. Here, all previous black-box models showed a complexity of $\Theta(n/\log n)$, which again is sightly smaller than what one would expect. The proofs of these results again heavily exploit that the oracle returns the precise fitness value. In spite of this, we still find a black-box algorithm in our ranking-based model that solves the problem with $\Theta(n/\log n)$ queries. We are currently not sure if this should be interpreted in the way that this function class is easier to optimize, or in the way that the ranking-based model still allows too powerful algorithms.

Though we claim that in this paper we introduce ranking-based black-box complexity, the work by Droste et al. [DJW06] implicitly contains a result on this as well. Droste et al. give a lower bound of $\Omega(n/\log(n))$ for the unrestricted black-box complexity of the class of all functions $g \circ f$, where $g : \mathbb{R} \to \mathbb{R}$ is strictly monotone increasing and f is a binary-value function. As we argue later, ranking-based complexity is the same as classical complexity after allowing arbitrary strictly monotone perturbations of the objective values. For this reason, Droste et al. implicitly show that the ranking-based black-box complexity of the binary-value functions is $\Omega(n/\log n)$, which we now improve to the sharp bound of $\Omega(n)$. Via the same argumentation, the conference version [DJTW03] of [DJW06] also shows that the ranking-based complexity of the leadingones class (defined in Section 6) is between $n/2 - o(n)$ and $n + 1$. Our lower-bound proof for the binary-value class also applies to this class, improving the lower bound from $n/2 - o(n)$ to $n - 2$, which is sharp apart from the additive term.

2 Notation and Previous Black-Box Models

In this section, we give a brief overview of two previous black-box models, the *unrestricted black-box model* by Droste, Jansen and Wegener [DJW06] and the more recent *unbiased black-box model* by Lehre and Witt [LW10].

Let us first fix the notations used frequently throughout the paper.

2.1 Notation

The positive integers are denoted by $\mathbb{N}$. For $k \in \mathbb{N}$, we abbreviate $[k] := \{1, \dots, k\}$. Similarly, we define $[0..k] := [k] \cup \{0\}$. For $k, \ell \in \mathbb{N}$ we write $[k \pm \ell] := [k - \ell, k + \ell] \cap \mathbb{Z}$.

Let $n \in \mathbb{N}$. For a bit string $x = x_1 \dots x_n \in \{0,1\}^n$, we denote by $\bar{x}$ the bitwise complement of x (i.e., for all $j \leq n$ we have $\bar{x}_j = 1 - x_j$).

If $x, y \in \{0,1\}^n$, we obtain the bit string $x \oplus y$ by setting, for each $j \in [n]$, $(x \oplus y)_i := 1$ if $x_i \neq y_i$ and $(x \oplus y)_i := 0$ if $x_i = y_i$. That is, $\oplus$ denotes the bitwise exclusive-or. We use the shorthand $|x|_1$ for the number of ones in the bit string x, i.e., $|x|_1 = \sum_{i=1}^n x_i$.

If f is a function and S a set, we write $f(S) := \{f(s)\,;\, s \in S\}$. We write id_S for the identity function of S, i.e., $\mathrm{id}_S(s) = s$ for all $s \in S$. For $n \in \mathbb{N}$, the set S_n contains all permutations of $[n]$. For $\sigma \in S_n$ and $x \in \{0,1\}^n$ we abbreviate $\sigma(x) := x_{\sigma(1)} \dots x_{\sigma(n)}$.

Lastly, we denote by log the natural logarithm to base $e := \exp(1)$. If we refer to a different base, we indicate this in the subscript, e.g., we write $\log_2$ for the binary logarithm.

All asymptotic notation (Landau symbols, big-Oh notation) will be with respect to n, which typically denotes the dimension of the search space $\{0,1\}^n$.

2.2 Unrestricted and Unbiased Black-Box Model

As mentioned in the introduction, we aim at continuing the development of a complexity theory for randomized search heuristics. Usually, the complexity of a problem is measured by the performance of the best algorithm out of some class of algorithms (e.g., all those algorithms which can be implemented on a Turing machine [GJ90], [Hro03]).

What distinguishes randomized search heuristics from classical algorithms is that they are problem-independent. As such, the only way they obtain information about the problem to be solved is by learning the objective value of possible solutions ("search points"). To ensure this problem-independence, one usually assumes that the objective function is given by an oracle or as a *black-box*. Using this oracle, the algorithm may query the objective value of all possible solutions, but any such query does only return this search point's objective value and no other information about the objective function.

For simplicity, we shall restrict ourselves to real-valued objective functions defined on the set $\{0,1\}^n$ of bit-strings of length n. This is motivated by the fact that many evolutionary algorithms use such a representation.

Naturally, we do allow that the algorithms use random decisions. From the black-box concept, it follows that the only type of action the algorithm may perform is, based on the objective values learned so far, deciding on a probability distribution on $\{0,1\}^n$, sampling a search point $x \in \{0,1\}^n$ according to this distribution, and querying its objective value from the oracle. This leads to the scheme of Algorithm 1, which we call an *unrestricted black-box algorithm*.

As performance measure of a black-box algorithm we take the number of queries to the oracle performed by the algorithm until it first queries an optimal solution. We call this the *run time*, or *optimization time*, of the black-box algorithm. This is justified by the observation that in typical applications of randomized search heuristics, the evaluation of the fitness of the search points is more costly than the generation of new search points. Since we are mainly talking about randomized algorithms, we regard the expected number of queries.

We can now follow the usual approach in complexity theory. Let $\mathcal{F}$ be a class of real-values functions defined on $\{0,1\}^n$. The *complexity* of an algorithm A for $\mathcal{F}$ is the maximum expected run time of A on a function $f \in \mathcal{F}$ (worst-case run time). The complexity of $\mathcal{F}$ with respect to a class $\mathcal{A}$ of algorithms is the minimum ("best") complexity among all $A \in \mathcal{A}$ for $\mathcal{F}$. The *unrestricted black-box complexity* of $\mathcal{F}$ is the complexity of $\mathcal{F}$ with respect to the class of all black-box algorithms. This is the black-box complexity as introduced by Droste, Jansen and Wegener [DJW06].

Algorithm 1. Scheme of an Unrestricted Black-Box Algorithm

```
1 Initialization: Sample $x^{(0)}$ according to some probability distribution $p^{(0)}$ on
  $\{0,1\}^n$. Query $f(x^{(0)})$.
2 Optimization: for $t = 1, 2, 3, \ldots$ until termination condition met do
3     Depending on $\big((x^{(0)}, f(x^{(0)})), \ldots, (x^{(t-1)}, f(x^{(t-1)})\big)$, choose a probability
      distribution $p^{(t)}$ on $\{0,1\}^n$.
4     Sample $x^{(t)}$ according to $p^{(t)}$, and query $f(x^{(t)})$.
```

It is easily seen that the class of all black-box algorithms is very powerful. For example, for any function class $\mathcal{F} = \{f\}$ consisting of one single function, the unrestricted black-box complexity of $\mathcal{F}$ is 1—the algorithm that simply queries an optimal solution of f as first action shows this bound.

These drawbacks of the unrestricted black-box model inspired Lehre and Witt [LW10] to introduce a more restrictive black-box model, where algorithms may generate new solution candidates only from random or previously generated search points and only by using unbiased operators. Still the model contains most of the commonly studied search heuristics, such as many $(\mu + \lambda)$ and (μ, λ) evolutionary algorithms, simulated annealing algorithms, the Metropolis algorithm, and the Random Local Search algorithm.

Definition 1 (k-ary unbiased variation operator). *Let $k \in \mathbb{N}$. A* k-ary unbiased distribution $D(. \mid y^{(1)}, \ldots, y^{(k)})_{y^{(1)},\ldots,y^{(k)} \in \{0,1\}^n}$ *is a family of probability distributions over $\{0,1\}^n$ such that for all* inputs $y^{(1)}, \ldots, y^{(k)} \in \{0,1\}^n$ *the following two conditions hold.*

$$(i) \forall x, z \in \{0,1\}^n : D(x \mid y^{(1)}, \ldots, y^{(k)}) = D(x \oplus z \mid y^{(1)} \oplus z, \ldots, y^{(k)} \oplus z) \text{ and}$$
$$(ii) \forall x \in \{0,1\}^n \, \forall \sigma \in S_n : D(x \mid y^{(1)}, \ldots, y^{(k)}) = D(\sigma(x) \mid \sigma(y^{(1)}), \ldots, \sigma(y^{(k)})) .$$

We refer to the first condition as $\oplus$-invariance *and to the second as* permutation invariance. *An operator sampling from a k-ary unbiased distribution is called a* k-ary unbiased variation operator.

Note that the only 0-ary unbiased distribution over $\{0,1\}^n$ is the uniform one. 1-ary, also called *unary* operators are sometimes referred to as mutation operators, in particular in the field of evolutionary computation. 2-ary, also called *binary* operators are often referred to as crossover operators. If we allow arbitrary arity, we call the corresponding model the $*$-ary unbiased black-box model.

k-ary unbiased black-box algorithms can now be described via the scheme of Algorithm 2. The *k-ary unbiased black-box complexity* of some class of functions $\mathcal{F}$ is the complexity of $\mathcal{F}$ with respect to all k-ary unbiased black-box algorithms.

Algorithm 2. Scheme of a k-ary Unbiased Black-Box Algorithm

```
1 Initialization: Sample $x^{(0)} \in \{0,1\}^n$ uniformly at random and query $f(x^{(0)})$.
2 Optimization: for $t = 1, 2, 3, \ldots$ until termination condition met do
3     Depending on $\big(f(x^{(0)}), \ldots, f(x^{(t-1)})\big)$ choose up to $k$ indices
      $i_1, \ldots, i_k \in [t-1]$ and a and a $k$-ary unbiased distribution
      $D(. \mid x^{(i_1)}, \ldots, x^{(i_k)})$.
4     Sample $x^{(t)}$ according to $D(. \mid x^{(i_1)}, \ldots, x^{(i_k)})$ and query $f(x^{(t)})$.
```

Note that for all $k \leq \ell$ each k-ary unbiased black-box algorithm is contained in the ℓ-ary unbiased black-box model.

Lehre and Witt [LW10] proved, among other results, that all functions with a single global optimum have a unary unbiased black-box complexity of $\Omega(n \log n)$. For several standard test problems this bound is met by different unary randomized search heuristics, such as the $(1+1)$ EA or the Random Local Search algorithm. Recall that, as pointed out above, the unrestricted black-box complexity of any such function is 1. For results on higher arity models refer to the work of Doerr et al. [DJK+10].

3 The Ranking-Based Black-Box Model

It has been commented by Hansen [Han10] that many standard randomized search heuristics do not take advantage of knowing the *exact* objective values.

Rather, they create new search points based on the *relative* objective values of previously queried search points. That is, after having queried t fitness values $f(x^{(1)}), \dots, f(x^{(t)})$, they rank the corresponding search points $x^{(1)}, \dots, x^{(t)}$ according to their relative fitness. The selection of input individuals $x^{(i_1)}, \dots, x^{(i_k)}$ for the next variation operator is based solely on this ranking. We define ranking as follows.

Definition 2. *Let S be a set, let $f : S \to \mathbb{R}$ be a function, and let C be a subset of S. The* ranking *ρ of C with respect to f assigns to each element $c \in C$ the number of elements in C with a smaller f-value plus 1, formally, $\rho(c) := 1 + |\{c' \in C\,;\, f(c') < f(c)\}|$.*

Note that two elements with the same f-value are assigned the same ranking.

As discussed above, many randomized search heuristics do only use the ranking of the search points seen so far. Therefore, we restrict the two black-box models which we introduced in the previous section to black-box algorithms that use no other information than this ranking.

Unrestricted Ranking-Based Black-Box Model. The unrestricted ranking-based black-box model can be described via the scheme of Algorithm 1 where we replace the third line by "Depending on the ranking of $\{x^{(0)} \dots, x^{(t-1)}\}$ with respect to f, choose a probability distribution $p^{(t)}$ on $\{0,1\}^n$."

Unbiased Ranking-Based Black-Box Model. For the definition of the unbiased ranking-based model we consider the scheme of Algorithm 2 and replace the third line by "Depending on the ranking of $\{x^{(0)} \dots, x^{(t-1)}\}$ with respect to f, choose up to k indices $i_1, \dots, i_k \in [t-1]$ and a k-ary unbiased distribution $D(. \mid x^{(i_1)}, \dots, x^{(i_k)})$."

Both ranking-based black-box models capture many common search heuristics, such evolutionary algorithms using elitist selection, ant colony optimization algorithms, and the Random Local Search algorithm. They do not include algorithms like simulated annealing algorithms, threshold accepting algorithms, or evolutionary algorithms using fitness proportional selection.

The ranking restriction in both the unrestricted and unbiased ranking-based model can equivalently be implemented by applying an unknown strictly monotone perturbation to the fitness values. That is, we assume there is a strictly monotone function[1] $g : \mathbb{R} \to \mathbb{R}$ unknown to the algorithm. Whenever the algorithm queries a search point x, the oracle only returns $g(f(x))$ (instead of the objective value $f(x)$). Again we adopt the worst-case view, that is, the performance of an algorithms is the worst among all $f \in \mathcal{F}$ and all strictly monotone functions $g : \mathbb{R} \to \mathbb{R}$.

We sometimes refer to this model as the (unrestricted or unbiased, respectively) *monotone* black-box model. In particular for upper bound proofs, this model is often more convenient to work with.

[1] Recall that a function $g : \mathbb{R} \to \mathbb{R}$ is said to be *strictly monotone* if for all $\alpha < \beta$ we have $g(\alpha) < g(\beta)$.

4 Ranking-Based Black-Box Complexity of OneMax

A classical easy test function in the theory of randomized search heuristics is the function $\textsc{OneMax}$, which simply counts the number of 1-bits, $\textsc{OneMax}(x) = \sum_{i=1}^{n} x_i$. The natural generalization of this particular function to a non-trivial class of functions is as follows.

Definition 3 (OneMax function class). *For $z \in \{0,1\}^n$ let $\mathrm{Om}_z : \{0,1\}^n \to [0..n], x \mapsto \mathrm{Om}_z(x) = |\{i \in [n]\,;\, x_i = z_i\}|$. The string $z = \arg\max \mathrm{Om}_z$ is called the* target string *of Om_z. Let $\textsc{OneMax}_n := \{\mathrm{Om}_z\,;\, z \in \{0,1\}^n\}$ be the set of all generalized $\textsc{OneMax}$ functions.*

In [DJK+10], the authors prove that the k-ary unbiased black-box complexity of $\textsc{OneMax}_n$ is $O(n/\log k)$. The following theorem shows that we can achieve the same bound in the (much weaker) unbiased ranking-based model.

Theorem 1. *For each $k \leq n$, the k-ary unbiased ranking-based black-box complexity of $\textsc{OneMax}_n$ is $O(n/\log k)$.*

For $k = \Theta(n)$ this statement is asymptotically optimal since already for the unrestricted black-box complexity a lower bound of $\Omega(n/\log n)$ has been shown by Anil and Wiegand [AW09].

The proof of Theorem 1 is quite technical. Due to space limitations, we only present a short overview here. Full proofs can be found in [DW11].

The main point of the proof is showing that the statement holds for $k = n$. We then generalize this case for arbitrary values of k. To this end, we derive from the case $k = n$ that one can independently optimize blocks of length k in $O(k/\log k)$ iterations, using k-ary unbiased variation operators only. Since there are $\lceil n/k \rceil$ such blocks of length k, the desired $O(n/\log k)$ bound follows by sequential optimization of these blocks.

The proof of the case $k = n$ again is divided into several steps. In the following description, all Greek symbols are constants and all statements hold with probability at least $1 - o(n^{-\lambda})$. We work in the monotone model, the unknown monotone perturbation is denoted by g, the unknown objective function by Om_z.

First, we show by simple Chernoff bound arguments that after drawing $\alpha n/\log n$ samples from $\{0,1\}^n$ uniformly at random, we have hit each fitness value $\ell \in [\frac{n}{2} \pm \kappa\sqrt{n}]$ at least once and that we have at least $\frac{1}{2}(1 - 2e^{-2\kappa^2})\alpha n \log^{-1} n$ samples x with objective value $\mathrm{Om}_z(x) \in [\frac{n}{2} \pm \kappa\sqrt{n}]$. Recall that we have not actually seen these objective values since they are hidden by the strictly monotone perturbation g.

Let x be a search point such that $g(\mathrm{Om}_z(x))$ is the median of the sampled objective values. Then $\mathrm{Om}_z(x)$, though unknown, lies in $[\frac{n}{2} \pm \log n]$. Consequently, $\mathrm{Om}_z(\bar{x})$ also lies in $[\frac{n}{2} \pm \log n]$. In particular, both $g(\mathrm{Om}_z(x))$ and $g(\mathrm{Om}_z(\bar{x}))$ lie in the interval $g\big([\frac{n}{2} \pm \kappa\sqrt{n}]\big)$, which is completely covered by the samples. Thus, a point y with $g(\mathrm{Om}_z(y))$ right in the middle of $g(\mathrm{Om}_z(x))$ and $g(\mathrm{Om}_z(\bar{x}))$ satisfies $\mathrm{Om}_z(y) = \frac{n}{2}$. Hence, by querying $g(\mathrm{Om}_z(\bar{x}))$ we exhibited one exact Om_z-value, which in turn tells us the exact Om_z-values of all samples x with $g(\mathrm{Om}_z(x)) \in g\big([\frac{n}{2} \pm \kappa\sqrt{n}]\big)$.

Lastly, we compute that there exists no $y \neq z$ such that $\text{OM}_z(x) = \text{OM}_y(z)$ for all samples x with $\text{OM}_z(x) \in [\frac{n}{2} \pm \kappa\sqrt{n}]$. In addition, the unique target string z can be created via an unbiased operation.

Putting everything together, we have defined an (unbiased) algorithm which identifies and creates the target string z in $\alpha n / \log n + 2$ iterations.

5 The Different Black-Box Complexities of BinaryValue

In the previous section, we have seen that the additional ranking restriction did not increase the black-box complexity of the OneMax functions class. In this section, we show an example where the two kinds of complexities greatly differ. Surprisingly, another simple class of classical test functions does the job, namely the class of generalized binary-value functions.

The binary-value function Bv is defined via $\text{Bv}(x) = \sum_{i=1}^{n} 2^{i-1} x_i$, that is, it assigns to each bit string its binary value. As before, we generalize this single function to a function classes BinaryValue_n (and BinaryValue_n^*), which are the $\oplus$-invariant ($\oplus$- and permutation-invariant) closure of the standard Bv function.

In the following we denote by δ the Kronecker symbol, i.e., for any two numbers $k, \ell \in \mathbb{N}_0$ we have $\delta(k, \ell) = 1$ if $k = \ell$ and $\delta(k, \ell) = 0$ otherwise.

Definition 4 (BinaryValue function class). *For $z \in \{0,1\}^n$ and $\sigma \in S_n$, we define the function $\text{Bv}_{z,\sigma} : \{0,1\}^n \to \mathbb{N}_0, x \mapsto \text{Bv}(\sigma(x \oplus \bar{z})) = \sum_{i=1}^{n} 2^{i-1} \delta(x_{\sigma(i)}, z_{\sigma(i)})$. We set $\text{Bv}_z := \text{Bv}_{z,\text{id}_{[n]}}$. We further define the classes $\text{BinaryValue}_n := \{\text{Bv}_z \,;\, z \in \{0,1\}^n\}$ and $\text{BinaryValue}_n^* := \{\text{Bv}_{z,\sigma} \,;\, z \in \{0,1\}^n, \sigma \in S_n\}$. If $f \in \text{BinaryValue}_n$ ($f \in \text{BinaryValue}_n^*$), there exist exactly one $z \in \{0,1\}^n$ (and exactly one $\sigma \in S_n$) such that $f = \text{Bv}_z$ ($f = \text{Bv}_{z,\sigma}$). Since $z = \arg\max \text{Bv}_z$ ($z = \arg\max \text{Bv}_{z,\sigma}$), we call z the* target string *of f. Similarly, we call σ the* target permutation *of $\text{Bv}_{z,\sigma}$.*

We show that the unbiased black-box complexity of BinaryValue_n^* is $O(\log n)$, cf. Theorem 2, whereas both ranking-based black-box complexities of BinaryValue_n are $\Theta(n)$, cf. Theorem 3.

Let us begin with the upper bound for the unbiased black-box complexity.

Theorem 2. *The $*$-ary unbiased black-box complexity of BinaryValue_n^* (and thus, the one of BinaryValue_n) is at most $\lceil \log_2 n \rceil + 2$.*

The main reason why the black-box complexity of BinaryValue_n^* is much lower than the one of OneMax_n is that each BinaryValue-function has 2^n different function values. Hence, each query reveals much more information about the underlying objective function. The complete proof can be found in [DW11].

We devote the remainder of this section to the proof of the following result.

Theorem 3. *The unrestricted ranking-based black-box complexity of BinaryValue_n and BinaryValue_n^* is larger than $n - 2$.*

As discussed in the introduction, Droste, Jansen, and Wegener [DJW06] implicitly showed a lower bound of $\Omega(n/\log n)$ for the setting of Theorem 3. Our lower bound of $n-2$ is almost tight. For the unrestricted ranking-based complexity, [DJW06] again implicitly provide an upper bound of $n+2$. For the unbiased case, previous work by Doerr et al. [DJK+10] immediately yields the following.

Corollary 1. *For all $k \geq 2$, the k-ary unbiased ranking-based black-box complexity of* $\textsc{BinaryValue}_n$ *and* $\textsc{BinaryValue}^*_n$ *is at most $2n$.*

To derive the lower bound in Theorem 3, we employ Yao's minimax principle [Yao77].

Theorem 4 (Yao's Minimax Principle, formulation following [MR95]). *Let Π be a problem with a finite set $\mathcal{I}$ of input instances (of a fixed size) permitting a finite set $\mathcal{A}$ of deterministic algorithms. Let p be a probability distribution over $\mathcal{I}$ and q be a probability distribution over $\mathcal{A}$. Then,*

$$\min_{A\in\mathcal{A}} \mathrm{E}[T(I_p, A)] \leq \max_{I\in\mathcal{I}} \mathrm{E}[T(I, A_q)]\,,$$

where I_p denotes a random input chosen from $\mathcal{I}$ according to p, A_q a random algorithm chosen from $\mathcal{A}$ according to q and $T(I,A)$ denotes the running time of algorithm A on input I.

We apply Yao's minimax principle in our setting as follows. We first show that in the ranking-based black-box model, any deterministic algorithm needs an expected number of at least $n-2$ iterations to optimize $\textsc{Bv}_z$, if $\textsc{Bv}_z$ is taken from $\textsc{BinaryValue}_n$ uniformly at random. Theorem 4 then implies that for any randomized algorithm A there exist at least one instance $\textsc{Bv}_z \in \textsc{BinaryValue}_n$ such that it takes, in expectation, at least $n-2$ iterations for algorithm A to optimize $\textsc{Bv}_z$. This implies Theorem 3.

The crucial observation is that when optimizing $\textsc{Bv}_z$ with a ranking-based algorithms, then from t samples we cannot learn more than $t-1$ bits of the hidden bit-string z. This is easy to see for two samples x, y. If $\textsc{Bv}_z(x) > \textsc{Bv}_z(y)$, we see that $x_k = z_k \neq y_k$, where $k := \max\{j \in [n]\,;\, x_j \neq y_j\}$, but all other bits of z can be arbitrary. That t samples do not reveal $\binom{t}{2}$ bits, but only $t-1$, is a consequence of the following combinatorial lemma (for the proof confer [DW11]).

Lemma 1. *Let $t \in [n]$ and let $x^{(1)}, \ldots, x^{(t)}$ be t pairwise different bit strings. For every pair $(i,j) \in [t]^2$ we set $\ell_{i,j} := \max\{k \in [n]\,;\, x^{(i)}_k \neq x^{(j)}_k\}$, the largest bit position in which $x^{(i)}$ and $x^{(j)}$ differ. Then $|\{\ell_{i,j}\,;\, i,j \in [t]\}| \leq t-1$.*

We are now ready to prove Theorem 3.

Proof (of Theorem 3). It is easily seen that the set $\mathcal{A}$ of all deterministic algorithms on $\textsc{BinaryValue}_n$ is finite, if we restrict our attention to those algorithms which stop querying search points after the n-th iteration.

As mentioned above, we equip $\textsc{BinaryValue}_n$ with the uniform distribution. Let $\textsc{Bv}_z \in \textsc{BinaryValue}_n$ be drawn uniformly at random and let $A \in \mathcal{A}$ be a (deterministic) algorithm. In the following, we show that prior to the t-th iteration, the set of still possible target strings has size at least 2^{n-t+1} and that all of these target strings have the same probability to be the desired target string **(A)**. Consequently, the probability to query the correct bit string in the t-th iteration, given that the algorithm has not found it in a previous iteration, is at most 2^{-n+t-1}. This yields that the expected number of iterations $\mathrm{E}[T(\textsc{Bv}_z, A)]$ until algorithm A queries the target string z can be bounded from below by

$$\begin{aligned}\sum_{i=1}^{n} i \cdot \Pr[A \text{ queries } z \text{ in the } i\text{-th iteration}] &\geq \sum_{i=1}^{n} i \cdot 2^{-n+i-1} \\ = \sum_{i=1}^{n} (n-i+1)\, 2^{-i} &= (n+1) \sum_{i=1}^{n} 2^{-i} - \sum_{i=1}^{n} i\, 2^{-i} \,. \end{aligned} \tag{1}$$

A simple, but nonetheless very helpful observation shows

$$\begin{aligned}\sum_{i=1}^{n} i\, 2^{-i} &= \sum_{i=1}^{n} 2^{-i} + \sum_{i=1}^{n} (i-1)\, 2^{-i} \\ &= (1-2^{-n}) + 2^{-1} \sum_{i=1}^{n} (i-1)\, 2^{-(i-1)} = (1-2^{-n}) + 2^{-1} \sum_{i=1}^{n-1} i\, 2^{-i} \,,\end{aligned}$$

yielding $\sum_{i=1}^{n} i\, 2^{-i} = 2(1-2^{-n}) - n2^{-n} = 2 - (n+2)2^{-n}$.

Plugging this into (1), we obtain

$$\mathrm{E}[T(\textsc{Bv}_z, A)] \geq (n+1)(1-2^{-n}) - (2-(n+2)2^{-n}) > n-2 \,.$$

This proves $\min_{A \in \mathcal{A}} \mathrm{E}[T(\textsc{Bv}_z, A)] > n-2$. Since any randomized ranking-based black-box algorithm $\tilde{A}$ can be modeled by randomly choosing a deterministic one A_q according to a suitable distribution q and then executing the latter, Yao's minimax principle (Theorem 4) implies $\max_{z \in \{0,1\}^n} \mathrm{E}[T(\textsc{Bv}_z, \tilde{A}_q)] \geq \min_{A \in \mathcal{A}} \mathrm{E}[T(\textsc{Bv}_z, A)] > n-2$. That is, the ranking-based black-box complexity of $\textsc{BinaryValue}_n$ is larger than $n-2$.

It remains to prove **(A)**. Let $t \leq n$ and let $x^{(1)}, \ldots, x^{(t)}$ be the search points which have been queried by the algorithm in the first t iterations. All the algorithm has learned about $x^{(1)}, \ldots, x^{(t)}$ is the ranking of these bit strings, i.e., it knows for all $i, j \in [t]$ whether $\textsc{Bv}_z\big(x^{(i)}\big) > \textsc{Bv}_z\big(x^{(j)}\big)$, or $\textsc{Bv}_z\big(x^{(i)}\big) < \textsc{Bv}_z\big(x^{(j)}\big)$, or $\textsc{Bv}_z\big(x^{(i)}\big) = \textsc{Bv}_z\big(x^{(j)}\big)$. Note that $\textsc{Bv}_z\big(x^{(i)}\big) = \textsc{Bv}_z\big(x^{(j)}\big)$ implies $x^{(i)} = x^{(j)}$. Thus, this case can be disregarded as one cannot learn any additional information by querying the same bit string twice.

As in Lemma 1 we set, for all $i, j \in [t]$, $\ell_{i,j} := \max\{k \in [n]\,;\, x^{(i)}_k \neq x^{(j)}_k\}$ and $\mathcal{L} := \{\ell_{i,j}\,;\, i, j \in [t]\}$.

Let $\ell \in \mathcal{L}$ and let $i, j \in [t]$ such that $\max\{k \in [n]\,;\, x^{(i)}_k \neq x^{(j)}_k\} = \ell$. We can fix $z_\ell = x^{(i)}_\ell$ if $\textsc{Bv}_z\big(x^{(i)}\big) > \textsc{Bv}_z\big(x^{(j)}\big)$, and we fix $z_\ell = x^{(j)}_\ell$ if $\textsc{Bv}_z\big(x^{(i)}\big) < \textsc{Bv}_z\big(x^{(j)}\big)$. That is, we can fix $|\mathcal{L}|$ bits of z.

Statement **(A)** follows from observing that for every single bit string z' with $z'_\ell = z_\ell$ for all $\ell \in \mathcal{L}$ the function $\text{Bv}_{z'}$ yields exactly the same ranking as Bv_z. Hence, all such z' are possible target strings. Since there is no way to differentiate between them, all of them are equally likely to be the desired target string.

Furthermore, it holds by Lemma 1 that $|\mathcal{L}| \leq t-1$. This shows that, at the end of the t-th iteration, there are at least $2^{n-(t-1)}$ possible target strings. It follows from Lemma 1 that either the algorithm has queried at most one of these possible target strings already, or $|\mathcal{L}| < t-1$. Consequently, prior to executing the $(t+1)$-st iteration, there are at most $2^{n-(t-1)} - 1 > 2^{n-t}$ bit strings which are equally likely to be the desired target string. This proves **(A)**. □

Note that already a much simpler proof, also applying Yao's minimax principle, shows the following general lower bound.

Theorem 5. *Let $\mathcal{F}$ be a class of functions such that each $f \in \mathcal{F}$ has a unique global optimum and such that for all $z \in \{0,1\}^n$ there exists a function $f_z \in \mathcal{F}$ with $z = \arg\max f_z$. Then the unrestricted ranking-based black-box complexity of $\mathcal{F}$ is $\Omega(n/\log n)$.*

6 Ranking-Based Black-Box Complexity of LeadingOnes

The proof ideas of the lower bound in the previous section can also be applied to another classical class of test functions called LeadingOnes. This closes a gap left open in [DJTW03].

For every bit string x, LeadingOnes(x) is defined to be the length of the longest prefix of ones. As before, we define two generalized classes of LeadingOnes functions.

Definition 5 (LeadingOnes function class). *Let $n \in \mathbb{N}$. For any $z \in \{0,1\}^n$ let $\text{Lo}_z : \{0,1\}^n \to \mathbb{N}, x \mapsto \max\{i \in [0..n]\,;\, x_i = z_i\}$, the length of the maximal joint prefix of x and z. Let LeadingOnes_n be the collection of all such functions, i.e., $\text{LeadingOnes}_n := \{\text{Lo}_z\,;\, z \in \{0,1\}^n\}$.*

For $z \in \{0,1\}^n$ and $\sigma \in S_n$ we set $\text{Lo}_{z,\sigma} : \{0,1\}^n \to \mathbb{N}, x \mapsto \max\{i \in [0..n]\,;\, x_{\sigma(i)} = z_{\sigma(i)}\}$, the maximal joint prefix of x and z with respect to σ. The set LeadingOnes^*_n contains all such functions, i.e., $\text{LeadingOnes}^*_n := \{\text{Lo}_{z,\sigma}\,;\, z \in \{0,1\}^n, \sigma \in S_n\}$.

As discussed in the last paragraph of the introduction, Droste et al. in the conference version [DJTW03] of paper [DJW06] implicitly show that the ranking-based black-box complexity of LeadingOnes_n is at least $\frac{n}{2} - o(n)$ and at most $n+1$. Using the same methods as for BinaryValue_n, we improve the lower bound to $n-2$ (noting that we did not try to optimize the additive gap of 3).

Theorem 6. *The unrestricted ranking-based black-box complexity of LeadingOnes_n (and thus, the unrestricted ranking-based black-box complexity of LeadingOnes^*_n) is strictly larger than $n-2$.*

Crucial for the proof is again the combinatorial statement of Lemma 1, from which we concluded that after t queries we know nothing more than $t-1$ bits of the hidden bit string z. We omit the proof details.

The following remark follows easily from the proof of Theorem 14 in [DJK+10].

Remark 1. *For every $k \geq 2$, the k-ary unbiased ranking-based black-box complexity of* LeadingOnes$^*_n$ *is $O(n \log n)$.*

7 Conclusions

Motivated by the fact that (i) previous complexity models for randomized search heuristics give unrealistic low complexities and (ii) that many randomized search heuristics only compare objective values, but not regard their absolute values, we added such a restriction to the two existing block-box models. While this does not change the black-box complexity of the OneMax function class (this remains relatively low at $\Theta(n/\log n)$), we do gain an advantage for the BinaryValue function class. Here the complexity is a ridiculous $O(\log n)$ without the ranking restriction, but $\Theta(n)$ in the ranking-based model. We obtain some more results improving previous work by different authors, summarized in Table 1. All results indicate that the ranking-based black-box complexity might be a promising measure for the hardness of problems for randomized search heuristics.

Table 1. Black-Box Complexity of OneMax, BinaryValue, and LeadingOnes. The lower bound for the unbiased ranking-based model also holds for the unrestricted ranking-based model. Abbreviations: unrestr. = unrestricted, ranking-b. = ranking-based.

Model	Arity	OneMax		BinaryValue		LeadingOnes	
unrestr.	n/a	$\Omega(n/\log n)$	[DJW06]	$2-2^{-n}$	[DJW06]	$\Theta(n)$	[DJW06]
		$O(n/\log n)$	[AW09]				
unbiased	1	$\Theta(n \log n)$	[LW10]	$\Theta(n \log n)$	[LW10]	$\Theta(n^2)$	[LW10]
	$2 \leq k \leq n$	$O(n/\log k)$	[DJK+10]	$O(n)$	[DJK+10]	$O(n \log n)$	[DJK+10]
	$*$			$O(\log n)$	(here)	$> n-2$	(here)
unbiased	$2 \leq k \leq n$	$O(n/\log k)$	(here)	$\Theta(n)$	(here)	$O(n \log n)$	(here)
ranking-b.	$*$					$> n-2$	(here)

References

[Ald83] Aldous, D.: Minimization algorithms and random walk on the d-cube. Annals of Probability 11, 403–413 (1983)

[AW09] Anil, G., Paul Wiegand, R.: Black-box search by elimination of fitness functions. In: Proc. of Foundations of Genetic Algorithms (FOGA 2009), pp. 67–78. ACM, New York (2009)

[DJK$^+$10] Doerr, B., Johannsen, D., Kötzing, T., Lehre, P.K., Wagner, M., Winzen, C.: Faster black-box algorithms through higher arity operators, ArXiv e-prints 1012.0952 (2010); Foundations of Genetic Algorithms XI FOGA (2011, to appear)

[DJTW03] Droste, S., Jansen, T., Tinnefeld, K., Wegener, I.: A new framework for the valuation of algorithms for black-box optimization. In: Proc. of Foundations of Genetic Algorithms (FOGA 2003), pp. 253–270 (2003)

[DJW06] Droste, S., Jansen, T., Wegener, I.: Upper and lower bounds for randomized search heuristics in black-box optimization. Theory of Computing Systems 39, 525–544 (2006)

[DW11] Doerr, B., Winzen, C.: Towards a Complexity Theory of Randomized Search Heuristics: Ranking-Based Black-Box Complexity, ArXiv e-prints 1102.1140 (2011)

[GJ90] Garey, M.R., Johnson, D.S.: Computers and intractability; a guide to the theory of np-completeness. W. H. Freeman & Co., New York (1990)

[Han10] Hansen, N.: Private communication (2010)

[HGD94] Horn, J., Goldberg, D., Deb, K.: Long path problems. In: Davidor, Y., Männer, R., Schwefel, H.-P. (eds.) PPSN 1994. LNCS, vol. 866, pp. 149–158. Springer, Heidelberg (1994)

[HK78] Hausmann, D., Korte, B.: Lower bounds on the worst-case complexity of some oracle algorithms. Discrete Math. 24, 261–276 (1978)

[Hro03] Hromkovič, J.: Algorithmics for hard problems: introduction to combinatorial optimization, randomization, approximation, and heuristics, 2nd edn. Springer, New York (2003)

[LTT89] Llewellyn, D.C., Tovey, C., Trick, M.: Local optimization on graphs. Discrete Appl. Math. 23, 157–178 (1989) Erratum: 46, 93–94 (1993)

[LW10] Lehre, P.K., Witt, C.: Black-box search by unbiased variation. In: Proc. of Genetic and Evolutionary Computation Conference (GECCO 2010), pp. 1441–1448. ACM, New York (2010)

[MR95] Motwani, R., Raghavan, P.: Randomized algorithms. Cambridge University Press, Cambridge (1995)

[Yao77] Yao, A.C.-C.: Probabilistic computations: Toward a unified measure of complexity. In: Proc. of 18th Annual Symposium on Foundations of Computer Science (FOCS 1977), pp. 222–227 (1977)

Learning Read-Constant Polynomials of Constant Degree Modulo Composites

Arkadev Chattopadhyay[1], Ricard Gavaldà[2], Kristoffer Arnsfelt Hansen[3], and Denis Thérien[4]

[1] University of Toronto
arkadev@cs.toronto.edu
[2] Universitat Politècnica de Catalunya
gavalda@lsi.upc.edu
[3] Aarhus University
arnsfelt@cs.au.dk
[4] McGill University
denis@cs.mcgill.ca

Abstract. Boolean functions that have constant degree polynomial representation over a fixed finite ring form a natural and strict subclass of the complexity class ACC^0. They are also precisely the functions computable efficiently by *programs* over fixed and finite nilpotent groups. This class is not known to be learnable in any reasonable learning model. In this paper, we provide a deterministic polynomial time algorithm for learning Boolean functions represented by polynomials of constant degree over arbitrary finite rings from membership queries, with the additional constraint that each variable in the target polynomial appears in a constant number of monomials. Our algorithm extends to superconstant but low degree polynomials and still runs in quasipolynomial time.

1 Introduction

Understanding the computational power of computation over rings of the form $\mathbb{Z}_m$, for an arbitrary composite number m, is a fundamental open problem. A concrete and natural setting in which to explore this power is the model of representing Boolean functions by low degree polynomials over such rings, in the following sense [4]: an assignment to the variables is a 1 of the Boolean function if and only if the polynomial on it evaluates to an element of a prespecified accepting subset of the ring.

When the modulus is a prime number and the ring thus turns into a finite field, our knowledge of representations is far better than the general case. For instance, it is known that degree $\Omega(n)$ is required in order to represent the Boolean function MOD_q by polynomials over the field $\mathbb{Z}_p$, when p is a prime and q has a prime factor different from p. The stronger result that MOD_q remains hard to even approximate well by such polynomials of low degree, is a key insight in the celebrated lower bound of Razborov [22] and Smolensky [24] on the size of bounded-depth circuits.

In contrast, we do not even know the exact degree of the Parity function for polynomials over $\mathbb{Z}_m$, as soon as m is an odd number having two distinct prime factors. In a beautiful work, Barrington, Beigel and Rudich [4] showed that composite moduli give non-trivial advantage to polynomials as compared to prime moduli. More precisely, they showed that the degree of the OR and the AND function over $\mathbb{Z}_m$ is $O\big(n^{1/t}\big)$ if m has t distinct prime factors. On the other hand, it is well known that if m is a fixed prime, then this degree is $\Omega(n)$. This surprising construction of Barrington *et al.* has found diverse applications. Indeed, Efremenko [14] recently built efficient locally decodable codes from it. Also, Gopalan [16] shows that several previously known constructions of explicit Ramsey graphs can all be derived from this construction.

The best known lower bounds on the composite degree of any Boolean function is $\Omega(\log n)$ (see for example [17,25,9] and the survey [13]). Proving anything better is a tantalizingly open problem. In this work, we look at low degree polynomials through the lens of computational learning theory. The motivation and hope is that this approach will lead to new insights into the structure of these polynomials, thus benefiting both the fields of learning theory and complexity theory.

Given that we know degree lower bounds of $\Omega(\log n)$, it is reasonable to hope that we can learn functions represented by constant degree polynomials. We take on this task in this paper in the setting where the learner is allowed to ask membership queries. The main difficulty that one faces is essentially the same that confronts one when proving lower bounds on the degree: while computation by the target polynomial takes place in the entire ring $\mathbb{Z}_m$, the information revealed to the learner is just Boolean. That is, we learn only whether the polynomial when evaluated on the chosen point yields an element of the unknown accepting set. Although several equivalent low degree representations may exist for the target concept, it is a non-trivial fact that polynomially many such queries are able to isolate a unique function in the concept class that agrees with the answers of the teacher. The computational challenge, of course, is to recognize this unique function.

Our Result. We consider the concept class of functions that have a representation by a constant degree polynomial in which every variable appears in a constant number of monomials. We show that this class is exactly learnable in polynomial time from the values of the target function at all input assignments of Hamming weight bounded by another constant. These values can be obtained, in particular, from membership queries. Additionally, our learning algorithm is proper in the sense that it outputs a constant degree polynomial equivalent to the target polynomial with respect to the Boolean function they compute. It is worth remarking that there are very few instances in which concepts are known to be properly learnable, especially when there is no guarantee of a unique representation.

Overview of Our Techniques. Our learning algorithm uses some novel ideas exploiting the following structural property of low degree polynomials first discovered in the work of Péladeau and Thérien [20] (see the translation [21]): for every constant degree polynomial P over any fixed finite commutative ring with identity, there exists a "magic set" of variables of constant cardinality such that every value in the range of P can be attained by only setting a subset of variables from the magic set to 1 and setting all other variables to 0. This property is very convenient and in particular, implies that every Boolean function that can be represented by a constant degree polynomial gets uniquely determined by the values it takes on points of constant Hamming weight. It is worthwhile to note that although the function gets fixed by knowing its behavior on all low weight points, it is not clear how to efficiently determine the value of this function on any other input point of the Boolean cube. This is the essential challenge that the learning algorithm has to overcome.

To be more specific, using this magic set we define an equivalence relation among monomials of the same degree. We show that there always exists a polynomial representing the same function that the teacher holds, in which all monomials belonging to the same equivalence class have identical coefficients. The number of equivalence classes is upper bounded by a constant and there is a very efficient test of equivalence. These properties allow us to enumerate all possible values of coefficients and then choose any that satisfies the polynomially many points of constant weight.

Relations to Existing Work. Polynomials have been widely studied in learning theory. When the learner can use evaluation queries returning the precise value of the polynomial over the base ring or field, polynomials of arbitrary degree over finite fields and even finite rings can be learned from evaluation+equivalence queries [23,8,12]. On the other hand, when the accepting set of the target polynomial is guaranteed to be a singleton set, it can be learned in the PAC model, and also approximately from membership queries alone, by a variation of the subspace-learning algorithm in [18] (see also [15]); this holds for all finite (and many infinite) rings. For the field $\mathbb{Z}_p$, a standard use of Fermat's little theorem shows that every polynomial of degree d with an arbitrary accepting set can be turned into an equivalent polynomial of degree $d(p-1)$ whose range is $\{0, 1\}$; this allows us to learn polynomials of constant degree over $\mathbb{Z}_p$ both in the PAC model, as above, and exactly from membership queries.

In this paper we make progress, for the first time to our best knowledge, in the equivalent learning problem for the non-field case. Note however that the problem was mentioned in [15], where the degree 1 case was solved by a technique that does not seem to extend to higher degrees. The emphasis in [15] was the classification of families of Boolean functions computed by *programs* over finite monoids (cf. [3,7,6]), with respect to their learnability in different models. In this setting, polynomials of constant degree over finite rings are equivalent in power to programs over nilpotent groups (as shown in [20]) with degree-1 polynomials corresponding to programs over Abelian groups. The class of functions computed by such programs is a natural subclass of functions computable by programs over solvable groups. Starting with the famous and surprising work of Barrington [3] that showed the class of functions computed by polynomial length programs over finite non-solvable groups is exactly the complexity class NC^1, programs over groups, or monoids in general, have been used (see for example [7,6]) to characterize natural subclasses of NC^1.

2 Preliminaries

2.1 Polynomials over Finite Rings

Let $\mathcal{R}$ be a commutative finite ring with unit, and let $P(x_1, \ldots, x_n)$ be a polynomial over $\mathcal{R}$. We say P is a *read-k* polynomial, if every variable in P appears in at most k monomials of P.

Consider a family of polynomials $\mathcal{P} = \{P_i\}_{i=1}^{\infty}$, where P_i is a polynomial in i variables. We say the family $\mathcal{P}$ is *read-constant*, if there exist a k such that every $P_i \in \mathcal{P}$ is read-k. Similarly, we say that $\mathcal{P}$ is *constant degree* if there exists d such that every $P_i \in \mathcal{P}$ is of degree at most d.

In this work, we will restrict our attention to variables ranging over the set $\{0, 1\} \subseteq \mathcal{R}$, and as a consequence we can without loss of generality restrict our attention to *multilinear* polynomials. Formally we consider the ring of polynomials $\mathcal{R}[x_1, \ldots, x_n]/N$, where N is the ideal generated by the set of polynomials $\{x_i^2 - x_i \mid i = 1, \ldots, n\}$. Any function $\{0, 1\}^n \rightarrow \mathcal{R}$ is uniquely expressed by such a polynomial. Define the *range* of P as $\mathrm{range}(P) = \{r \in \mathcal{R} \mid \exists x \in \{0, 1\}^n : P(x) = r\}$.

Equipping a polynomial P with an *accepting set* $A \subseteq \mathcal{R}$, we say that the pair (P, A) computes a Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ if it holds that $P(x) \in A$ if and only if $f(x) = 1$, for all $x \in \{0, 1\}^n$.

Given a set of indices $J \subseteq [n]$, we let $\chi_J \in \{0, 1\}^n$ denote the characteristic vector of J. Conversely, for $w \in \{0, 1\}^n$, define $I_w = \{i \in [n]\} \mid w_i = 1\}$. Thus $\chi_{I_w} = w$ and $I_{\chi_J} = J$. For $u, v \in \{0, 1\}^n$, let $u \vee v \in \{0, 1\}^n$ be defined by $I_{u \vee v} = I_u \cup I_v$.

Consider now a degree d polynomial, $P(x) = \sum_{I \subset [n], |I| \leq d} c_I \prod_{i \in I} x_i$. For a subset $S \subseteq [n]$ we define the polynomial P_S of monomials from S by, $P_S(x) = \sum_{I \subseteq S, |I| \leq d} c_I \prod_{i \in I} x_i$. For disjoint subsets $S, T \subseteq [n]$ define the polynomial $P_{S \times T}$ consisting of cross terms between S and T:

$$P_{S \times T}(x) = \sum_{I, J \neq \emptyset; I \subseteq S, J \subseteq T; |I| + |J| \leq d} c_{I \vee J} \prod_{i \in I \cup J} x_i \ .$$

For a polynomial we associate the graph G_P defined as follows. The set of vertices of G_p is $\{1, \ldots, n\}$ and the set of edges is $E(G_P) = \{(i, j) \mid x_i$ and x_j appear together in some monomial of P $\}$ This will allow us to speak of the *distance* between variables of P, namely as distances in the graph G_P.

2.2 Structural Properties of Polynomials

Using an inductive Ramsey-theoretic argument, the following important structural result about constant degree polynomials over finite rings was proved by Péladeau and Thérien [20].

Theorem 1 (Péladeau and Thérien). *Let $\mathcal{R}$ be a finite commutative ring with unity and let d be any number. Then there exists a constant $c = c(\mathcal{R}, d)$ with the following property: For any multilinear polynomial P over $\mathcal{R}$ of degree at most d and for any $r \in \text{range}(P)$ there exists $w \in \{0,1\}^n$ with $|I_w| \leq c$ such that $P(w) = r$.*

Remark 2.
– The theorem as stated above is actually only implicitly given in the proof of Lemma 2 of [20].
– In Sect. 4 we shall present with full proof a quantitative strengthening of the theorem based on a result of Tardos and Barrington [25].

Two easy consequences of this theorem are given below. Our learning algorithm will be heavily based on these results.

Corollary 3. *There exists a constant $s = s(\mathcal{R}, d)$, such that for every multilinear polynomial P over $\mathcal{R}$ of degree at most d, there exists a set $J \subset \{1, \ldots, n\}$ with the following properties: (1). $|J| \leq s$. (2). For every $r \in \text{range}(P)$ there exists $w \in \{0,1\}^n$ with $I_w \subseteq J$ such that $P(w) = r$.*

Proof. Let $s = |\mathcal{R}|c(\mathcal{R}, d)$, with $c(\mathcal{R}, d)$ as given by Theorem 1. We can then simply take J to be the union of $|\text{range}(P)|$ sets I_w provided by Theorem 1 for each $r \in \text{range}(P)$. □

For a given polynomial P we will refer to the set J as guaranteed above to exist as the *magic set* set of variables for P.

Corollary 4. *There exists a constant $c' = c'(\mathcal{R}, d)$ with the following property: Let P and Q be polynomials of degree at most d with accepting sets A and B, respectively. If the Boolean functions computed by the pairs (P, A) and (Q, B) agree on all inputs $w \in \{0,1\}^n$ with $|I_w| \leq c'$, then the two Boolean functions are identical.*

Proof. We take $c' = c(\mathcal{R} \times \mathcal{R}, d)$ as given by Theorem 1. Now, write $P(x) = \sum c_I \prod_{i \in I} x_i$ and $Q(x) = \sum d_I \prod_{i \in I} x_i$. Consider the polynomial $(P \times Q)$ over $\mathcal{R} \times \mathcal{R}$ given by $(P \times Q)(x) = \sum (c_I, d_I) \prod_{i \in I} x_i$ If (P, A) and (Q, B) do not compute the same Boolean function there is $(r, s) \in \text{range}(P \times Q)$ such that either $r \in A$ and $s \notin B$ or $r \notin A$ and $s \in B$. Then by Theorem 1 and the choice of c' this would be witnessed by a $w \in \{0,1\}^n$ with $|I_w| \leq c'$ such that $(P \times Q)(w) = (r, s)$. □

3 Learning with Membership Queries

In this section we will present our algorithm for learning read-constant, constant degree polynomials. For convenience we choose to present the algorithm as a nondeterministic algorithm, that when terminating with success always output a correct polynomial. Afterwards we will be able to convert this nondeterministic algorithm into a deterministic algorithm simply by enumerating over all possible sequences of guesses of the algorithm, arguing that there are only polynomially many such sequences.

For ensuring that the nondeterministic algorithm always produces a correct output we use a consistency check procedure, described as Algorithm 1.

Algorithm 1. Consistent(Q, A, f)

```
Input: Polynomial $Q$ with accepting set $A \subseteq \mathbb{Z}_m$. Membership query
       access to Boolean function $f$.
Output: Decides if the pair $(Q, A)$ computes the function $f$.
1: Query $f$ on all $w \in \{0,1\}^n$ with $|I_w| \le c'(\mathcal{R}, d)$
2: Return true if and only if for each queried $w$, $f(w) = 1$ if and only if
   $Q(w) \in A$
```

The correctness of the procedure is immediate from Corollary 4.

3.1 Equivalence Relations between Monomials

For our algorithm we need the following somewhat technical definition of parameterized equivalence relations of monomials. Intuitively, they serve the following purpose: we want to learn an unknown polynomial, singling it out from exponentially many possibilities. One way to reduce this huge search space is to deduce, from membership queries, that some of the $n^{O(d)}$ coefficients must be the same, as they can only have a constant ($|R|$) number of values. Equivalence among two monomials, as defined below, is intended to suggest that they define isomorphic subpolynomials of the target polynomial.

For example, if a target polynomial contains terms $2x_1$, $3x_2$, x_1x_2, $2x_3$, $3x_4$, x_3x_4 we would like to say that monomials x_1x_2 and x_3x_4 are equivalent, and when searching for coefficients for these monomials we can discard all settings of coefficients where they differ.

The idea of the learning algorithm, to be explained in more detail in the next section, is to first implement equivalent tests among all monomials and then, on the basis of this information, actually find the values of all coefficients exploring a polynomial search space rather than an exponential one.

For any monomial M, let $I_M = \{i \mid x_i \text{ appears in } M\}$. Conversely, for any set of indices I let M_I denote the monomial $\Pi_{i\in I} x_i$.

Given a polynomial P and a set J of indices in $\{1, \dots, n\}$, we define a parameterized equivalence relation $\sim_{d,J}$ on tuples $(M, \preceq)$, where $M \subset [n], M \cap J = \emptyset$, $|M| = d$, and $\preceq$ is a total ordering[1] on $[n]$, by induction on d. We say $(M, \preceq_1) \sim_{d,J} (M', \preceq_2)$ if the following is satisfied:

1. For every assignment w, such that $I_w \subseteq J$, we have $P(w \vee \chi_M) \in A$ if and only if $P(w \vee \chi_{M'}) \in A$.

[1] Alternatively one could fix the same ordering, say $1, \dots, n$ for all monomials. However we find it natural to identify monomials that that are identical up to a permutation of the variables.

2. Let $M_1, \ldots, M_d$ (and $M'_1, \ldots, M'_d$) be the subsets of M (M') of size $d-1$ listed in the lexicographic order w.r.t $\preceq_1$ $(\preceq_2)$. Then $(M_i, \preceq_1) \sim_{d-1,J} (M'_i, \preceq_2)$, for all $i \leq d$.

For every pair of monomials M and M', each of degree d, with $I_M, I_{M'}$ disjoint from J, we say that $M \equiv_{d,J} M'$ if there exist $\preceq_1$ and $\preceq_2$ such that $(I_M, \preceq_1) \sim_{d,J} (I_{M'}, \preceq_2)$.

3.2 Idea of the Learning Algorithm

Let P be a polynomial with accepting set A. Let $\mathcal{M}_{\text{range}(P)}$ be the subgroup of the additive subgroup of $\mathcal{R}$ generated by range(P) (i.e. the $\mathbb{Z}$-module generated by range(P)). Consider next the following equivalence relation on monomials of a polynomial P

For tuples $(M, \preceq_1)$ and $(M', \preceq_2)$, where $M, M' \subset [n]$ and $\preceq_1$ and $\preceq_2$ are total orderings, we say $(M, \preceq_1) \widehat{\sim}_d (M', \preceq_2)$ if the following is satisfied:

1. For every $r \in \mathcal{M}_{\text{range}(P)}$, $r + P(\chi_M) \in A$ if and only if $r + P(\chi_{M'}) \in A$.
2. Let $M_1, \ldots, M_d$ (and $M'_1, \ldots, M'_d$) be the subsets of M (M') of size $d-1$ listed in the lexicographic order w.r.t $\preceq_1$ $(\preceq_2)$. Then $(M_i, \preceq_1) \widehat{\sim}_{d-1} (M'_i, \preceq_2)$, for all $i \leq d$.

For every pair of monomials M and M', each of degree d, we say that $M \widehat{\equiv}_d M'$ if there exist $\preceq_1$ and $\preceq_2$ such that $(I_M, \preceq_1) \widehat{\sim}_d (I_{M'}, \preceq_2)$.

Assume now (by induction) that in P, that for all monomial M_1 and M_2 of degree $s < r$ we have $c_{M_1} = c_{M_2}$, whenever $M_1 \widehat{\equiv}_s M_2$. Next consider monomials M and M' with $M \widehat{\equiv}_r M'$, and let P' be the polynomial obtained from P by replacing the coefficient of M' with the coefficient of M. By our (inductive) assumption we have $P(\chi_M) = P'(\chi_{M'})$. Let $x \in \{0,1\}^n$ be arbitrary. Now, since $P(x), P(\chi_{M'}) \in \text{range}(P)$ we have $r = P(x) - P(\chi_{M'}) \in \mathcal{M}_{\text{range}(P)}$. We then have $r + P(\chi_M) \in A$ if and only if $r + P(\chi_{M'}) \in A$. But $r + P(\chi_{M'}) = (P(x) - P(\chi_{M'})) + P(\chi_{M'}) = P(x)$ and $r + P(\chi_M) = (P(x) - P(\chi_{M'})) + P(\chi_M) = P(x) - P(\chi_{M'}) + P'(\chi_{M'}) = P'(x)$. Hence $P(x) \in A$ if and only if $P'(x) \in A$.

Thus, if we would be able to actually *implement* testing of the above equivalence relation, we would be have a simple learning algorithm as follows: First compute all equivalence classes. Then enumerate all candidate polynomials obtained by all possible coefficients for these equivalence classes, and test for correctness using the **Consistent** procedure. We do not know how to accomplish this. However using the notion of a magic set, we *are* in fact able to implement (possibly a refinement of) this equivalence relation on P restricted to *all but a constant number of variables.*

3.3 Properties of Polynomials Equipped with a Magic Set

Before stating our learning algorithm, we establish a number of properties to be used later for polynomials P equipped with a magic set J.

Lemma 5. *Let $P(x) = \sum_{I \subseteq [n], |I| \leq d} c_I \prod_{i \in I} x_i$ be any polynomial over $\mathcal{R}$, with a magic set J. Let N be the set of indices that (viewed as vertices in the graph G_P) are at distance at least 2 from J in G_P. Then, $r + \sum_{I \subseteq N, |I| \leq d} \lambda_I c_I \in \text{range}(P)$, for all $r \in \text{range}(P)$ and all $\lambda_I \in \{0, \ldots, |\mathcal{R}| - 1\}$.*

Proof. We prove the statement by induction, first by induction on the degree d, and then by further induction on the monomials of degree d. We take as our induction hypothesis that $r + \sum_{I \subseteq N, |I| < d} \lambda_I c_I \in \text{range}(P)$ for all r and λ_I. The base case $d = 0$ trivially holds. Consider now for the inductive step the case $d + 1$. Enumerate all $\binom{|N|}{d}$ subsets of N of cardinality d, and let I^i denote the ith set in this enumeration. We shall now further induct on k to show that $r + \sum_{I \subseteq N, |I| < d} \lambda_I c_I + \sum_{i=1}^{k} \lambda_{I^i} c_{I^i} \in \text{range}(P)$ for every $r \in \text{range}(P)$. The $k = 0$ base case trivially holds. For the inductive step, we want to show that $r + \sum_{I \subseteq N, |I| < d} \lambda_I c_I + \sum_{i=1}^{k} \lambda_{I^i} c_{I^i} + \lambda c_{I^{k+1}} \in \text{range}(P)$. By induction hypothesis, there exists u with $I_u \subset J$ such that $P(u) = r + \sum_{I \subset L, |I| < d} \lambda_I c_I + \sum_{i=1}^{k} \lambda_{I^i} c_{I^i} = r_0$. Then clearly, $P(u \vee \chi_{I^{k+1}}) = r_0 + \sum_{I \subset I^{k+1}} c_I + c_{I^{k+1}} = r_1 \in \text{range}(P)$. Hence, there is u_1 with $I_{u_1} \subseteq J$ such that $P(u_1) = r_1$. Continuing in this way for λ times we see that $r_\lambda = r_0 + \lambda \cdot (\sum_{I \subset I^{k+1}} c_I) + \lambda c_{I^{k+1}} \in \text{range}(P)$. Applying once more our outer induction hypothesis, we conclude $r_\lambda + (|\mathcal{R}| - \lambda)(\sum_{I \subset I^{k+1}} c_I) = r_0 + \lambda \cdot c_{I^i} \in \text{range}(P)$. This completes the inner and outer induction. □

For a polynomial P with accepting set A we can always obtain equivalent polynomial in which the constant term is 0 by *shifting* the accepting set according to the constant term. Thus in the following assume the constant term of P is 0. Let J be a magic set of P. Let N be the set of indices that are at distance 2 or more from J in G_P. Let P_N be the polynomial obtained from P by fixing to 0 every variable indexed in the set $[n] \setminus N$.

The crucial insight required for limiting the amount of nondeterministic guesses in our learning algorithm is expressed in the following lemma.

Lemma 6. *Let P be any polynomial of degree d with accepting set A and a magic set J, and let $r \leq d$. Assume that for all monomials M_1 and M_2 in P_N of degree $s < r$ we have $c_{M_1} = c_{M_2}$ whenever $M_1 \equiv_{s,J} M_2$. Consider now monomials M and M' of degree r in P_N such that $M \equiv_{r,J} M'$. Let P' be the polynomial obtained from P by replacing the coefficient of M' with the coefficient of M. Then the polynomials P and P' compute the same Boolean function.*

Proof. Since $M \equiv_{r,J} M'$, we have $(I_M, \preceq_M) \sim_{r,J} (I_{M'}, \preceq_{M'})$ for some $\preceq_M$ and $\preceq_{M'}$. Let us enumerate lexicographically the subsets of I_M and I_N according to $\preceq_M$ and $\preceq_{M'}$. Let M_i and M'_i be the monomial corresponding to the ith such subsets and let d_i denote their degree. By definition we have $M_i \equiv_{d_i,J} M'_i$, and so by assumption the coefficients of M_i and M'_i are the same. We thus have that $P(\chi_M) = P'(\chi_{M'})$.

To prove that P and P' with accepting set A compute the same Boolean function, let $x \in \{0,1\}^n$ be arbitrary. Obviously, $P(x) \in \text{range}(P)$. By Lemma 5 we then have that also $P(x) - P(\chi_{M'}) \in \text{range}(P)$. It follows there exist u with

$I_u \subseteq J$ such that $P(u) = P(x) - P(\chi_{M'})$. Since $M \equiv_{r,J} M'$ we have $P(u \vee \chi_M) = P(u) + P(\chi_M) \in A$ if and only if $P(u \vee \chi_{M'}) = P(u) + P(\chi_{M'}) \in A$. But $P(u) + P(\chi_M) = P(x) - P(\chi_{M'}) + P(\chi_M) = P(x) - P(\chi_{M'}) + P'(\chi_{M'}) = P'(x)$ and $P(u) + P(\chi_{M'}) = P(x)$. Hence we can conclude that $P(x) \in A$ if and only if $P'(x) \in A$. □

3.4 The Learning Algorithm

We are now finally in position to state our algorithm.

Algorithm 2. Learn-Poly(f)

```
   Input: Membership query access to Boolean function $f$.
   Output: Returns pair $(Q, A)$ computing the function $f$, or fail.
1: Nondeterministically guess the following:
     A magic set $J \subseteq [n]$, $|J| \leq s(\mathcal{R}, d)$ and polynomial $Q_J$.
     The set $K \subseteq [n]$ at distance 1 in $G_P$ from $J$ and polynomials $Q_K$ and $Q_{K \times J}$.
     The set $L \subseteq [n]$ at distance 2 in $G_P$ from $J$, and polynomial $Q_{K \times L}$.
     An accepting set $A \subseteq \mathcal{R}$ for $Q$.
2: Let $N = [n] \setminus (J \cup K)$.
3: Query $f$ on all inputs $(w \vee x)$ where $I_w \subseteq J$, $I_x \subseteq [n] \setminus N$, and $|I_x| \leq 2d$.
4: Compute the equivalence classes of $\equiv_{r,J}$, for all $r = 1, \ldots, d$, over
   monomials $M_I$ with $I \subseteq N$ and $|I| \leq d$.
5: Nondeterministically guess an element of $\mathcal{R}$ for each equivalence class.
6: Construct polynomial
   $Q = Q_J + Q_K + Q_{J \times K} + Q_{K \times L} + \sum_{I \subset N, |I| \leq d} c_I \cdot M_I$, where $c_I \in \mathcal{R}$ is the
   element guessed for the equivalence class of monomial $M_I$.
7: If Consistent$(Q, A, f)$, output $(Q, A)$, otherwise output fail.
```

Theorem 7. *Let $\mathcal{R}$ be a fixed commutative finite ring with unit. Let F be the class of Boolean functions that can be computed by read-constant and constant degree polynomials.* **Learn-poly** *non-deterministically learns exactly any function $f \in F$ in polynomial time.*

Proof. Take a computation path of **Learn-poly** in which it made right guesses in step 1. Then using Lemma 6, we maintain an equivalent polynomial if the guesses for coefficients of each equivalence class is correct. If incorrect guesses result in a wrong candidate polynomial it will be detected by the **Consistent** procedure. □

It is easy to see that a deterministic variant of **Learn-poly** can be derived and it runs in poly-time. This can be done by simply going through all possible guesses. Since cardinality of J is bounded by a constant (using Corollary 3) determined by the degree of the polynomial, there are only polynomially many sets to guess. Since P is read-k for some constant k, $|K| \leq k(d-1)|J|$, and the number of guesses for K is at most $\binom{n}{k(d-1)|J|}$, which is again polynomial in n. Observe that the size of K is also bounded by a constant. Thus, guessing $P_J, P_K, P_{J\times K}$ involves at most $|\mathcal{R}|^s$ guesses, where s is the number of monomials of degree at most d involving variables indexed by set $K \cup J$. A similar argument shows that polynomially many guesses are needed to get the correct L for each possible K and then constantly many guesses for a given K and L are involved for $P_{K\times L}$. Since the equivalence relation $\equiv_{d,J}$ is finite indexed for each d, constantly many guesses have to be enumerated to also make this deterministic and we are done.

4 Extensions to Higher Degrees

For a Boolean function f on n variables, define $\Delta(f,\mathcal{R})$ to be the minimal degree of a polynomial over $\mathcal{R}$ computing f. Consider a family of Boolean functions $f = \{f_n\}_{n=1}^{\infty}$, one for each input length. Define $\Delta(f,\mathcal{R},n) = \Delta(f_n,\mathcal{R})$. Define $\Lambda(f,\mathcal{R},d)$ as the maximal n such that $\Delta(f,\mathcal{R},n) \leq d$.

The notion of the degree of the Boolean AND function allows the following quantitative version of Theorem 1.

Proposition 8. $c(\mathcal{R},d) \leq \Lambda(\text{AND},\mathcal{R},d)$

Proof. Let P be a multilinear polynomial of degree d over $\mathcal{R}$ in n variables. Let $r \in \text{range}(P)$. We will find $w \in \{0,1\}^n$ with $|I_w| \leq \Lambda(\text{AND},\mathcal{R},d)$ such that $P(w) = r$. If $P(0) = r$, we are done. Otherwise, pick $w \in \{0,1\}^n$ such that $|I_w|$ is minimal with $P(w) = r$. Consider now the restriction P' of P to the variables indexed by I_w. By minimality of $|I_w|$, we have that P' computes the AND function with accepting set $\{r\}$ on $|I_w|$ variables. Thus it follows that $|I_w| \leq \Lambda(\text{AND},\mathcal{R},d)$. □

As a consequence we obtain the following bounds for $s(\mathcal{R},d)$-the size of the magic set for polynomials over $\mathcal{R}$ of degree d, and $c'(\mathcal{R},d)$-the Hamming weight of assignments that uniquely identify a Boolean function represented by such a polynomial, in terms of $\Lambda(\text{AND},\mathcal{R},d)$ as well, following the proofs of Corollary 3 and Corollary 4.

Corollary 9. *$s(\mathcal{R},d) \leq |\mathcal{R}|\Lambda(\text{AND},\mathcal{R},d)$ and $c'(\mathcal{R},d) \leq \Lambda(\text{AND},\mathcal{R}\times\mathcal{R},d)$.*

Thus lower bounds for the degree of the AND function implies upper bounds on the above quantities. The degree of the AND function have been intensively studied over the ring $\mathbb{Z}_m$ [4,25]. Let in the following $m = p_1^{k_1}\cdots p_r^{k_r}$ have r distinct prime factors, and let $q_{\min} = \min(p_1^{k_1},\ldots,p_r^{k_r})$ and $q_{\max} = \max(p_1^{k_1},\ldots,p_r^{k_r})$. With these definitions, Tardos and Barrington [25] obtained the following lower bound, which is currently the best known.

Theorem 10 (Tardos and Barrington)
$\Delta(\text{AND},\mathbb{Z}_m,n) \geq ((1/(q_{\min}-1) - o(1))\log n)^{1/(r-1)}$.
Equivalently, $\Lambda(\text{AND},\mathbb{Z}_m,d) \leq 2^{(q_{\min}-1+o(1))d^{r-1}}$

For the purpose of our learning algorithm we are interested in bounds for the ring $\mathcal{R} = \mathbb{Z}_m^l$. In fact, without loss of generality we may assume that $\mathcal{R}$ is of this form. We can transfer the above results to this ring using standard methods. Let $m' = p_1\cdots p_r$. Let $p_{\min} = \min(p_1,\ldots,p_r)$.

Lemma 11. $\Delta(\text{AND}, \mathbb{Z}_{m'}, n) \leq l(q_{\max} - 1)\Delta(\text{AND}, \mathbb{Z}_m^l, n)$.
Equivalently, $\Lambda(\text{AND}, \mathbb{Z}_m^l, d) \leq \Lambda(\text{AND}, \mathbb{Z}_{m'}, l(q_{\max} - 1)d)$.

Proof. Let P be a polynomial over $\mathbb{Z}_m^l$ of degree d computing the AND function. Without loss of generality, the accepting set is $\{0\}$. Let $P_1, \ldots, P_l$ be the l coordinate polynomials. Consider a fixed j, and the polynomials $P_1, \ldots, P_l$ modulo $p_j^{k_j}$. By well known arguments (see e.g [25]) we can find polynomials $Q_1^j, \ldots, Q_l^j$ of degree at most $(p_j^{k_j} - 1)d$ such that $Q_i^j(x) \equiv 0 \pmod{p_j}$ if and only if $P_i(x) \equiv 0 \pmod{p_j^{k_j}}$, and furthermore $(Q_i^j(x) \mod p_j) \in \{0, 1\}$ for all x.

Define $Q^j(x) = 1 - \prod_{i=1}^{l}(1 - Q_i^j(x))$. We then have $Q^j(x) \equiv 0 \pmod{p_j}$ if and only if $P_i(x) \equiv 0 \pmod{p_j^{k_j}}$ for all x and i. Note the degree of Q^j is at most $l(p_j^{k_j} - 1)d$.

Considering all such polynomials, $Q^1, \ldots, Q^l$, from the Chinese Remainder Theorem we may find a polynomial Q, of degree at most $l(q_{\max} - 1)d$ such that $Q(x) \equiv 0 \pmod{m'}$ if and only if $P(x) = 0$ for all x. □

Combining Proposition 8, Theorem 10, and Lemma 11 we obtain the following concrete bounds (following the proofs of Corollaries 3 and 4):

Proposition 12

$$c(\mathbb{Z}_m^l, d) \leq \Lambda(\text{AND}, \mathbb{Z}_m^l, d) \leq 2^{(p_{\min}-1+o(1))(l(q_{\max}-1)d)^{r-1}} .$$
$$s(\mathbb{Z}_m^l, d) \leq |\mathbb{Z}_m^l| c(\mathbb{Z}_m^l, d) \leq m^l 2^{(p_{\min}-1+o(1))(l(q_{\max}-1)d)^{r-1}} .$$
$$c'(\mathbb{Z}_m^l, d) \leq c(\mathbb{Z}_m^{2l}, d) \leq 2^{(p_{\min}-1+o(1))(2l(q_{\max}-1)d)^{r-1}} .$$

We now provide a brief analysis of the running time of the deterministic version of our algorithm **Learn-Poly**, presented in the last section, in terms of parameters $s(\mathcal{R}, d)$ and $c'(\mathcal{R}, d)$. The algorithm asks membership queries on points of Hamming weight at most $c'(\mathcal{R}, d) + 2d$. Thus, $O(n^{c'+2d})$ many membership queries are asked in total. The algorithm runs over all possible choices of a magic set J of size $s = s(\mathcal{R}, d)$, the set K, of size $ks(d-1)$, of variables that are at distance 1 from the set J and set L, of size $k^2 s(d-1)^2$, at distance 2 from J. Thus the total number of such choices is at most $n^{k^2 s d^2}$. For each such choice of J, K, L, the algorithm considers all possible degree d polynomials of variables indexed in $J \cup K \cup L$. Thus, it has to consider at most $|\mathcal{R}|^{d(k^2 s d^2)^d}$ many polynomials. Further, for each choice of J, K, L it does equivalence testing for monomials that are free of variables indexed by J or K. There are at most dn^d such monomials and the test for each involves 2^s assignments of variables in J. Thus, equivalence testing takes $O\big(dn^d \cdot 2^s\big)$ time. It is not hard to see that degree d monomials split up into at most $|\mathcal{R}|^{2^d}$ equivalence classes. Thus, one has to consider all possible ways of coloring equivalence classes with elements of $\mathcal{R}$ giving rise to $|\mathcal{R}|^{d|\mathcal{R}|^{2^d}}$ such choices. Finally having guessed an entire candidate polynomial, the algorithm invokes procedure **Consistent** that verifies the consistency of the polynomial with all weight $c' = c'(\mathcal{R}, d)$ assignments. This requires $O\big(n^d \cdot cn^{c'+1}\big)$ time. Summing these up, the total running time is

$$O(2^{|\mathcal{R}|}) \times O(n^{O(k^2sd^2)}) \times O(|\mathcal{R}|^{d(k^2sd^2)^d}) \times O(dn^d 2^s) \times O(|\mathcal{R}|^{d|\mathcal{R}|^{2^d}}) \times O\big(n^d \cdot cn^{c'+1}\big)$$

Using Proposition 12, we see that for each $\mathcal{R} = \mathbb{Z}_m^\ell$ with a fixed m and ℓ, there exists a constant γ such that $s(\mathcal{R}, d), c'(\mathcal{R}, d) \leq \gamma^{d^{r-1}}$, where r is the number of distinct prime factors of m. Hence, combining the above observations we get the following:

Theorem 13. *Let m and ℓ be any fixed positive numbers. The class of Boolean functions representable by read-k polynomials of degree d over $\mathbb{Z}_m^\ell$ are exactly learnable from membership queries by a deterministic algorithm of running time $O\big(n^{k^2\gamma^{d^r}d^2} \times \gamma^{k^{2d}\gamma^{d^r}} \times \gamma^{\gamma^{2^d}}\big)$, where $\gamma = \gamma(m, \ell)$ is a constant and r is the number of distinct prime factors of m.*

Theorem 13 gives us a range of super-constant k and d for which we get sub-exponential running time. For instance, if we choose $k = o(\log \log n)$, and $d = o(\log \log \log n)$, the running time is $n^{(\log n)^{o(1)}}$.

5 Future Work

While the progress we make is limited from a learning theory perspective, the combinatorics involved is unexpectedly delicate, and suggests some further questions in understanding the structure of polynomials over rings of the form $\mathbb{Z}_m$.

The obvious next question is to remove the read-constant restriction in our result. Read-constant restrictions have been used, on several occasions, both in complexity theory and in learning theory. For example in complexity theory, Barrington and Straubing [5] proved superlinear bounds on the length of read-constant branching programs of bounded-width. Very recently, several works have been concerned with constructing pseudorandom generators for read-once branching programs of small width [10,11,19]. In learning theory, read-constant conditions have been sometimes shown to be unavoidable for efficient learning. For example, read-once Boolean formulas can be learned efficiently from membership and equivalence queries [2]. On the other hand, under cryptographic assumptions, even read-thrice Boolean formulas are impossible to learn no matter what polynomially-evaluatable hypothesis class is used (i.e., hard to learn in a representation-independent way).

In other cases, read-constant conditions for learning a target concept class can be removed at the expense of moving to a larger hypothesis class, which bypasses some computational bottleneck. For example, Aizenstein *et al.* [1] showed that read-k, satisfy-j DNF formulas[2] are learnable (as DNF formulas). Without the read-k condition, satisfy-j DNF formulas are not known to be learnable as

[2] A DNF formula is read-k if every variable appears at most k times; a DNF formula is satisfy-j if no assignment satisfies more than j terms simultaneously.

DNF, but they can be learned as Multiplicity Automata, as pointed out in [8]. Analogously, it is possible that constant degree polynomials over finite rings can be learned (in some reasonable learning model) by not insisting that the output is itself a constant degree polynomial.

Acknowledgments. A. Chattopadhyay is partially supported by a Natural Sciences and Engineering Research Council (NSERC) postdoctoral fellowship and research grants of Prof. T. Pitassi. R. Gavaldà is partially funded by the Spanish Ministry of Science and Technology contract TIN-2008-06582-C03-01 (SESAAME), by the Generalitat de Catalunya 2009-SGR-1428 (LARCA), and by the EU PASCAL2 Network of Excellence (FP7-ICT-216886).

References

1. Aizenstein, H., Blum, A., Khardon, R., Kushilevitz, E., Pitt, L., Roth, D.: On learning read-k-satisfy-j dnf. SIAM J. Comput. 27(6), 1515–1530 (1998)
2. Angluin, D., Hellerstein, L., Karpinski, M.: Learning read-once formulas with queries. J. ACM 40(1), 185–210 (1993)
3. Barrington, D.A.M.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. J. Comput. Syst. Sci. 38(1), 150–164 (1989)
4. Barrington, D.A.M., Beigel, R., Rudich, S.: Representing boolean functions as polynomials modulo composite numbers. Comput. Complexity 4, 367–382 (1994)
5. Barrington, D.A.M., Straubing, H.: Superlinear lower bounds for bounded-width branching programs. J. Comput. Syst. Sci. 50(3), 374–381 (1995)
6. Barrington, D.A.M., Straubing, H., Thérien, D.: Non-uniform automata over groups. Inform. and Comput. 89(2), 109–132 (1990)
7. Barrington, D.A.M., Thérien, D.: Finite monoids and the finite structure of NC^1. J. ACM 35(4), 941–952 (1988)
8. Beimel, A., Bergadano, F., Bshouty, N., Kushilevitz, E., Varricchio, S.: Learning functions represented as multiplicity automata. J. ACM 47, 506–530 (2000)
9. Bourgain, J.: Estimation of certain exponential sums arising in complexity theory. C. R. Acad. Sci. Paris 340(9), 627–631 (2005)
10. Braverman, M., Rao, A., Raz, R., Yehudayoff, A.: Pseudorandom generators for regular branching programs. In: 51th Annual IEEE Symposium on Foundations of Computer Science, pp. 40–47. IEEE Computer Society Press, Los Alamitos (2010)
11. Brody, J., Verbin, E.: The coin problem, and pseudorandomness for branching programs. In: 51th Annual IEEE Symposium on Foundations of Computer Science, pp. 30–39. IEEE Computer Society Press, Los Alamitos (2010)
12. Bshouty, N., Tamon, C., Wilson, D.: Learning matrix functions over rings. Algorithmica 22, 91–111 (1998)
13. Chattopadhyay, A.: Multilinear polynomials modulo composites. Bulletin of the European Association on Theoretical Computer Science, Computational Complexity Column, 100 (February 2010)
14. Efremenko, K.: 3-query locally decodable codes of subexponential length. In: 41st Annual Symposium on Theory of Computing, pp. 39–44. ACM Press, New York (2009)
15. Gavaldà, R., Thérien, D.: An algebraic perspective on boolean function learning. In: Gavaldà, R., Lugosi, G., Zeugmann, T., Zilles, S. (eds.) ALT 2009. LNCS, vol. 5809, pp. 201–215. Springer, Heidelberg (2009)

16. Gopalan, P.: Constructing ramsey graphs from boolean function representations. In: 21st Annual IEEE Conference on Computational Complexity, pp. 115–128. IEEE Computer Society Press, Los Alamitos (2006)
17. Grolmusz, V.: On the weak mod m representation of boolean functions. Chicago J. Theoret. Comput. Sci. (1995)
18. Helmbold, D.P., Sloan, R.H., Warmuth, M.K.: Learning nested differences of intersection-closed concept classes. Machine Learning 5, 165–196 (1990)
19. Koucký, M., Nimbhorkar, P., Pudlák, P.: Pseudorandom generators for group products. In: 43rd Annual ACM Symposium on Theory of Computing. ACM Press, New York (2011, to appear)
20. Péladeau, P., Thérien, D.: Sur les langages reconnus par des groupes nilpotents. C. R. Math. Acad. Sci. Paris Sér I Math. 306(2), 93–95 (1988)
21. Péladeau, P., Thérien, D.: On the languages recognized by nilpotent groups (a translation of "sur les languages reconnus par des groupes nilpotents"). In: Electronic Colloquium on Computational Complexity (ECCC), vol. 8(40) (2001)
22. Razborov, A.A.: Lower bounds on the size of bounded-depth networks over a complete basis with logical addition. Math. Notes of the Acad. of Sci. of USSR 41(3), 333–338 (1987)
23. Schapire, R.E., Sellie, L.: Learning sparse multivariate polynomials over a field with queries and counterexamples. J. Comput. Syst. Sci. 52(2), 201–213 (1996)
24. Smolensky, R.: Algebraic methods in the theory of lower bounds for boolean circuit complexity. In: 19th Annual ACM Symposium on Theory of Computing, pp. 77–82. ACM Press, New York (1987)
25. Tardos, G., Barrington, D.A.M.: A lower bound on the MOD-6 degree of the OR function. Comput. Complexity 7(2), 99–108 (1998)

On the Arithmetic Complexity of Euler Function

Manindra Agrawal[⋆]

IIT Kanpur
manindra@iitk.ac.in

Abstract. It is shown that the problem of computing the Euler function is closely related to the problem of computing the permanent of a matrix as well as to the derandomization of the Identity Testing problem. Specifically, it is shown that (1) if computing the Euler function over a finite field is hard then computing permanent over the integers is also hard, and (2) if computing any factor of the Euler function over a field is hard then the Identity Testing problem over the field can be derandomized.

1 Introduction

Leonhard Euler defined the following function, called the *Euler function*, in the course of his investigations into partition numbers:

$$E(x) = \prod_{n>0} (1 - x^n).$$

This function is the reciprocal of generating function of partition numbers:

$$\frac{1}{E(x)} = \sum_{m \geq 0} p_m x^m,$$

where p_m is the mth partition number (the number of ways of expressing m as a sum of positive numbers). Function $E(x)$ has many remarkable properties (some shown by Euler and others later), some of which we list below.

- $E(x)$ can be written as an infinite sum:

  $$E(x) = \sum_{n=-\infty}^{\infty} (-1)^n x^{\frac{1}{2}n(3n-1)}.$$

  This is called the *Pentagonal Number Theorem*, and can be proved in many ways [4].
- Roots of $E(x)$ are precisely all the roots of unity and each root of $E(x)$ has unbounded multiplicity. Therefore, for any root of unity ω, for any $m \geq 1$, the following holds [5]:

  $$\sum_{n=-\infty}^{\infty} (-1)^n [\frac{1}{2}n(3n-1)]^m (\omega)^{\frac{1}{2}n(3n-1)} = 0.$$

[⋆] N Rama Rao professor, Department of Computer Science, IIT Kanpur, Kanpur 208016, India.

– $E(x)$ exhibits strong symmetries, captured by *modular group*, and is closely related to *modular forms* [3].

We are interested in computing the Euler function. Of course, since it is an infinite series, we will aim to compute approximations of $E(x)$. There are two natural ways of defining approximations of $E(x)$:

Product approximation. Let $E_{\Pi,n}(x) = \prod_{m=1}^{n}(1 - x^m)$. Thus, $E_{\Pi,n}(x)$ is a polynomial of degree $\frac{1}{2}(n^2 + n)$ and $\lim_{n \mapsto \infty} E_{\Pi,n}(x) = E(x)$.

Sum approximation. Let $E_{\Sigma,n}(x) = \sum_{m=-n}^{n}(-1)^m x^{\frac{1}{2}m(3m-1)}$. Thus, $E_{\Sigma,n}(x)$ is a polynomial of degree at most $\frac{1}{2}(3n^2 + n)$ and $\lim_{n \mapsto \infty} E_{\Sigma,n}(x) = E(x)$.

The polynomial families $\{E_{\Sigma,n}(x)\}_{n>0}$ and $\{E_{\Pi,n}(x)\}_{n>0}$ can be computed by families of circuits of finite size. For computing polynomials, the natural model to adopt is that of arithmetic circuits, in which each gate computes addition or multiplication of its inputs. As these are univariate polynomials, the input to any such circuit will be the variable x and some constants (constants make the circuit family non-uniform). The *size* of such a circuit is defined as the number of wires in it. As any arithmetic circuit of size s with input x and constants can compute a polynomial of degree at most 2^s, it follows that one needs circuits of size $\Omega(\log n)$ to compute $E_{\Sigma,n}(x)$ or $E_{\Pi,n}(x)$. So the question we address is the following: can the two polynomial families be computed by circuit families of size $\log^{O(1)} n$? We show that a negative answer to any of these questions will separate VP from VNP (the arithmetic analogs of classes P and NP, see [8]). Moreover, a stronger lower bound on the complexity of $\{E_{\Pi,n}(x)\}_{n>0}$ will imply a complete black-box derandomization of Polynomial Identity Testing (PIT) problem, one of the central problems in complexity theory.

Pascal Koiran [6] has recently shown similar results as above. His results are applicable to a larger class of univariate polynomials, but do not imply as strong lower bounds as ours.

2 Arithmetic Complexity Classes and Permanent Polynomial

Arithmetic circuits over a ring R contains addition and multiplication gates (subtraction is simulated by multiplying with -1 and adding). We allow these gates to have unbounded fanin. The input gates are labelled by variables or constants from R. Such a circuit computes a polynomial in $R[z_1, \ldots, z_n]$, where $z_1, \ldots, z_n$ are the input variables.

Valient [8] defined VP, the class of multivariate polynomials that can be computed by small sized arithmetic circuits. To define this class, we need the notion of *formal degree*:

1. The formal degree of an input gate is equal to 1.
2. The formal degree of an addition gate is the maximum of the formal degrees of its incoming gates, and the formal degree of a multiplication gate is the sum of the formal degrees of incoming gates.

Finally, the *formal degree of a circuit* is the formal degree of its output gate. Clearly, this is an upper bound on the degree of the polynomial computed by the circuit. We now define the class VP_F:

Definition 2.1. *Polynomial family $\{P_n\}_{n>0}$ over field F belongs to the class VP_F if there exists a polynomial $p(n)$ and a sequence $\{C_n\}_{n>0}$ of arithmetic circuits over F such that C_n computes P_n and the size and formal degree of C_n at most $p(n)$.*

The constraint on the formal degree ensures that polynomials of high degree such as e.g. z^{2^n} and large constants such as 2^{2^n} cannot be computed. VP contains several important families of polynomials, e.g., the family of determinant polynomials.

Note that the definition of the class VP_F is tailored for multivariate polynomials. For univariate polynomial families, we do not consider the question if they belong to VP_F or not; instead, we simply consider their size (without bothering about the formal degree).

Another important complexity class is the class VNP. Polynomials in this family are sums over exponentially many values of a polynomial in VP. Formally,

Definition 2.2. *Polynomial family $\{P_n(x_1, \ldots, x_{u(n)})\}_{n>0}$ is in the class VNP_F if there exists a polynomial family $\{Q_n(x_1, \ldots, x_{v(n)})\}_{n>0}$, $v(n) = u^{O(1)}(n)$, in VP_F such that:*

$$P_n(x_1, \ldots, x_{u(n)}) = \sum_{\overline{\epsilon} \in \{0,1\}^{v(n)-u(n)}} Q_n(x_1, \ldots, x_{u(n)}, \overline{\epsilon}).$$

The family of permanent polynomials is in the class VNP_F for any field F. Moreover, Valient [8] showed that the permanent family is complete for the class VNP_F for characteristic $\neq 2$:

Theorem 2.3. *Let F be any field of characteristic $\neq 2$. For every family $\{P_n\}_{n>0}$ in VNP_F there exists a polynomially bounded function $p(n)$ and a matrix M of size $p(n)$ whose entries are variables and constants such that the family $2^{p(n)} P_n = \mathrm{per}(M)$.*

The factor $2^{p(n)}$ can be removed if F is of finite characteristic. The central question in the arithmetic complexity theory is whether $\mathrm{VP}_F \neq \mathrm{VNP}_F$. This is equivalent, by the above theorem, to the question whether the permanent family belongs to VP_F when char $F \neq 2$.

We will make use of the following lemma proved by Valiant [8] (the formulation below is by Koiran [7]).

Lemma 2.4. *Let F be a field of characteristic p. Suppose that $n \mapsto p(n)$ is a polynomially bounded function, and that $f : \mathbb{N} \times \mathbb{N} \to F$ is such that the map $1^n 0 j \mapsto f(j,n)$ is in the complexity class $\mathrm{Mod}_p\mathrm{P/poly}$ ($\mathrm{GapP/poly}$ for $p = 0$). Then the family $\{P_n\}_{n>0}$ of multilinear polynomials defined by*

$$P_n(x_1, \ldots, x_{p(n)}) = \sum_{j \in \{0,1\}^{p(n)}} f(j,n) x_1^{j[1]} \cdots x_{p(n)}^{j[p(n)]} \quad (1)$$

is in VNP_F. Here $j[k]$ denotes the $(k-1)^{th}$ least significant bit in the binary expansion of j.

3 The Family $\{E_{\Sigma,n}(x)\}_{n>0}$

For $\{E_{\Sigma,n}(x)\}_{n>0}$, it is easy the show the following theorem:

Theorem 3.1. *Let F be any field of characteristic > 2 and $s(\cdot)$ a monotonically nondecreasing function with $s(n) = \Omega(n)$. If the permanent family can be computed by arithmetic circuits of size $s(n)$ over F then the family $\{E_{\Sigma,n}(x)\}_{n>0}$ can be computed by arithmetic circuits of size $s(O(\log n))$ over F.*

Proof. For the polynomial $E_{\Sigma,n}(x)$, define its multilinear version as:

$$E_{\Sigma,M,n}(z_1, \ldots, z_u) = \sum_{t=0}^{\frac{1}{2}(3n^2+n)} c_t \prod_{j=1}^{u} z_j^{t[u]},$$

where $u = \lceil \log(3n^2 + n) \rceil - 1$ and c_t equals 0, 1, or -1 satisfying the condition that $E_{\Sigma,n}(x) = E_{\Sigma,M,n}(x, x^2, x^4, \ldots, x^{2^u})$. The coefficients c_t is computable in P given t: check if t is of the form $\frac{1}{2}(3m^2 \pm m)$, if yes, the coefficient is ± 1 depending on the sign on m, otherwise the coefficient is 0. Lemma 2.4 and Theorem 2.3 imply that $2^{p(n)} E_{\Sigma,M,n}(z_1, \ldots, z_u)$ can be computed as permanent of a size $\log^{O(1)} n$ matrix over F. A careful examination of the proofs of above results yields a matrix of size $O(\log n)$. By our assumption on the complexity of permanent, this polynomial can be computed by a circuit C of size $s(O(\log n))$ over F. The inputs to circuit C are variables z_1, …, z_u. Modify C by substituting x^{2^j} for variable z_j and inserting a small circuit computing x^{2^j} from x (this will contain j multiplication gates). This circuit now computes the polynomial $2^{p(n)} E_{\Sigma,n}(x)$ and has size $s(O(\log n))$. Multiply the output of the circuit with the multiplicative inverse of $2^{p(n)}$ (since the characteristic of the field is > 2, the inverse always exists). The resulting circuit computes the polynomial $E_{\Sigma,n}(x)$. □

4 The Family $\{E_{\Pi,n}(x)\}_{n>0}$

This is a more interesting class of polynomials. The polynomial $E_{\Pi,n}(x)$ agrees with $E_{\Sigma,n}(n)$ up to degree n, however, computing $E_{\Pi,n}(x)$ is far trickier because its coefficients are not as nice as for $E_{\Sigma,n}(x)$.

The interest in this family also derives from a conjecture made in [1] implying that computing $E_{\Sigma,n}(x)$ requires circuits of size $n^{\Omega(1)}$ over any field F. If the conjecture is true, what can we say about the complexity of permanent? If an analog of Theorem 3.1 holds for the family $\{E_{\Pi,n}(x)\}_{n>0}$ then we get that permanent requires circuits of size $2^{\Omega(n)}$ over F. A result of [6] gives a weaker conclusion: permanent requires circuits of size superpolynomial over integers (this follows from the fact shown in [6] that the coefficients of $E_{\Pi,n}(x)$ are computable in the counting hierarchy).

In this section, we prove a stronger version of result in [6]:

Theorem 4.1. *Let F be any field of characteristic $p > 2$ and $s(\cdot)$ a monotonically nondecreasing function with $s(n) = \Omega(n)$. If the permanent family can be computed by arithmetic circuits of size $s(n)$ over $\mathbb{Z}$ then the polynomial family $\{E_{\Pi,n}(x)\}_{n>0}$ can be computed by arithmetic circuits of size $s(s(O(\log n)))$ over F.*

In the remainder of this section, we prove this theorem.

Let $P(x) = E_{\Pi,n}(x)$ for some $n > 1$. The degree of $P(x)$ equals $d = \frac{1}{2}n(n+1) < n^2$. Since coefficients of $P(x)$ lie in F_p, computing $P(x)$ over F is equivalent to computing it over F_p. Let $\hat{F}$ be the smallest extension of F_p of size $\geq n^2$, and $q = |\hat{F}|$. So, by the Langrange's interpolation formula, we have:

$$P(x) = \sum_{\alpha \in \hat{F}} P(\alpha) \frac{\prod_{\beta \in \hat{F}, \beta \neq \alpha}(x - \beta)}{\prod_{\beta \in \hat{F}, \beta \neq \alpha}(\alpha - \beta)}$$

Observe that

$$\prod_{\beta \in \hat{F}, \beta \neq \alpha} (\alpha - \beta) = \prod_{\beta \in \hat{F}^*} \beta = -1,$$

and

$$\begin{aligned}\prod_{\beta \in \hat{F}, \beta \neq \alpha} (x - \beta) &= \frac{\prod_{\beta \in \hat{F}}(x - \beta)}{x - \alpha} \\ &= \frac{x^q - x}{x - \alpha} \\ &= \sum_{j=1}^{q-1} \alpha^{j-1} x^{q-j}.\end{aligned}$$

Therefore,

$$\begin{aligned}P(x) &= -\sum_{\alpha \in \hat{F}} P(\alpha) \sum_{j=1}^{q-1} \alpha^{j-1} x^{q-j} \\ &= -\sum_{j=1}^{q-1} (\sum_{\alpha \in \hat{F}} P(\alpha) \alpha^{j-1}) x^{q-j}.\end{aligned}$$

Now the coefficients of $P(x)$ are in the "right" form, and we can compute them efficiently if we can compute $P(\alpha)$ easily. So we turn our attention to

$$P(\alpha) = \prod_{m=1}^{n} (1 - \alpha^m).$$

A direct computation of $P(\alpha)$ is not possible as it needs a large number of multiplications. Instead, we exploit the fact that the computation is over $\hat{F}$ to show the following:

Lemma 4.2. *The coefficient $P(\alpha)$, given α, can be computed in* $\mathrm{P}^{\#\mathrm{P}}$.

Proof. Let $\gamma \in \hat{F}$ be a generator of the group $\hat{F}^*$. Define an NTM M as follows:

> On input α, guess t and m with $0 \leq t < q$ and $1 \leq m \leq n$. Check if $g^t = 1 - \alpha^m$. If not, then output 0 on the path. If yes, then output t on the path.

NTM M works for polynomial time (on input size $|\alpha| = O(\log n)$). We also have $\#M(\alpha) = \sum_{m=1}^{n} t_m$ where $g^{t_m} = 1 - \alpha^m$. Hence, $g^{\#M(\alpha)} = P(\alpha)$. Therefore, $P(\alpha)$ is computable in $\mathrm{P}^{\#\mathrm{P}}$. □

The rest of the proof is now straightforward. By the above lemma, it follows that the polynomial $P(x)$ can be computed in $\mathrm{VNP}_F^{\#\mathrm{P}}$. Since, by assumption, the permanent family can be computed by arithmetic circuits of size $s(n)$ over $\mathbb{Z}$, we get, by the completeness of permanent as argued in the previous section, that $P(x)$ can be computed in VNP_F with an advice of size $s(O(\log n))$. Applying the completeness of permanent once again, we get that $P(x)$ can be computed by circuits of size $s(O(s(O(\log n)))) = s(s(O(\log n)))$ over F.

5 Black-Box Derandomization of PIT

Result in the previous section suggests that computing $\{E_{\Pi,n}(x)\}_{n>0}$ may be hard. Is it easier to compute polynomials that are multiples of polynomials in $\{E_{\Pi,n}(x)\}_{n>0}$? A negative answer to this question yields a black-box derandomization of PIT.

Definition 5.1. *The* Polynomial Identity Testing (PIT) over field F *has as input an arithmetic circuit of size n over n variables and field F. The problem is to determine if the circuit computes an identically zero polynomial.*

We also classify polynomial families that are multiples of $\{E_{\Pi,n}(x)\}_{n>0}$:

Definition 5.2. *Let $\{P_n(x)\}_{n>0}$ be a family of polynomials with $P_n(x)$ of degree $n^{O(1)}$. The family is an m-*multiple of $\{E_{\Sigma,n}(x)\}_{n>0}$ *if for every n, $E_{\Pi,m}(x)$ divides $P_n(x)$.*

It is believable that $E_{\Pi,m}(x)$ requires circuits of size m^δ for some $\delta > 0$. Does this also mean that a non-zero $P_n(x)$ requires circuits of size bigger than m^δ? If yes, then we can obtain a black-box derandomization of PIT.

Theorem 5.3. *Suppose that any family $\{P_n(x)\}_{n>0}$ of non-zero polynomials over field F that is an m-multiple of $\{E_{\Sigma,n}(x)\}_{n>0}$ requires circuits of size m^δ over F to compute for some $\delta > 0$. Then there exists a polynomial-time black-box derandomization of PIT over F.*

Proof. Let C be a circuit of size n computing a polynomial $Q(y_1, \ldots, y_n)$ over field F. The degree of Q is bounded by 2^n. Let $D = 2^n + 1$. Replace y_i by $x^{D^{i-1}}$ for $1 \le i \le n$ as input to C. To compute these powers of y, insert additional multiplication gates at the bottom of the circuit C – one needs $O(n^2)$ additional gates. Let the resulting circuit be $\hat{C}$ and let $R(x)$ be the polynomial computed by $\hat{C}$. The size of $\hat{C}$ is $O(n^2)$ and the degree of $R(x)$ is bounded by 2^{n^2}. It is easy to observe that Q is identically zero if and only if R is. Test if $R(x) = 0$ modulo $(x^\ell - 1)^k$ for $1 \le \ell \le m = n^{\frac{3}{\delta}}$ and k the largest number such that $(x^\ell - 1)^k$ divides $E_{\Pi,m}(x)$, and output ZERO iff all the tests succeed.

The above algorithm is clearly a deterministic, polynomial-time, black-box algorithm. We now prove that it is also correct. Observe that if $R(x) = 0$ modulo $(x^\ell - 1)^k$

for every ℓ, $1 \leq \ell \leq m$ implies that $E_{\Pi,m}(x)$ divides $R(x)$. If $R(x)$ is non-zero then, by our assumption, $R(x)$ requires a circuit of size $m^\delta = n^3$ to compute over F. However, $\hat{C}$ is a circuit of size $O(n^2)$ computing R. Therefore, if $R(x) = 0$ modulo each of $(x^\ell - 1)^k$, then $R(x) = 0$. □

A special case of the above theorem was shown in [2]: the family of polynomials

$$P_n(x) = \prod_{a=1}^{\log^{\frac{7}{2}} n} ((x+a)^n - x^n - a),$$

can be computed by circuits of size $O(\log^{\frac{9}{2}} n)$ and any non-zero $P_n(x)$ over Z_n is not a $(\log^5 n)$-multiple of $\{E_{\Pi,n}(x)\}_{n>0}$.

6 Open Questions

A number of questions remain unanswered:

1. Is the polynomial $E_{\Pi,n}(x)$ over F_p computable in Mod$_p$P? We conjecture yes.
2. Does $E_{\Pi,n}(x)$ require circuits of size n^δ for some $\delta > 0$? We conjecture yes. Coupled with the above conjecture, this will imply a lower bound of $2^{\Omega(n)}$ on the size of arithmetic circuits computing permanent of size n matrices.
3. Does any m-multiple of $\{E_{\Pi,n}(x)\}_{n>0}$ require circuits of size m^δ for some $\delta > 0$? We conjecture yes. This will, in addition to the lower bounds, provide a derandomization of PIT.

References

[1] Agrawal, M.: Proving lower bounds via pseudo-random generators. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 92–105. Springer, Heidelberg (2005)
[2] Agrawal, M., Kayal, N., Saxena, N.: PRIMES is in P. Annals of Mathematics 160(2), 781–793 (2004)
[3] Apostol, T.M.: Modular Functions and Dirichlet Series in Number Theory. Springer, Heidelberg (1990)
[4] Bell, J.: Euler and the pentagonal number theorem, http://arxiv.org/abs/math.HO/0510054
[5] Euler, L.: De mirabilis proprietatibus numerorum pentagonalium. Acta Academiae Scientarum Imperialis Petropolitinae 4, 56–75 (1783),Translation by Jordan Bell, http://arxiv.org/abs/math/0505373
[6] Koiran, P.: Shallow circuits with high-powered inputs, http://arxiv.org/abs/1004.4960
[7] Koiran, P.: Valiants's model and the cost of computing integers. Computational Complexity 13, 131–146 (2004)
[8] Valiant, L.G.: Completeness classes in algebra. In: Proceedings of Annual ACM Symposium on the Theory of Computing, pp. 249–261 (1979)

Pseudo-random Graphs and Bit Probe Schemes with One-Sided Error[*]

Andrei Romashchenko

CNRS, LIF de Marseille & IITP of RAS (Moscow)

Abstract. We study probabilistic bit-probe schemes for the membership problem. Given a set A of at most n elements from the universe of size m we organize such a structure that queries of type "$x \in A$?" can be answered very quickly.

H. Buhrman, P.B. Miltersen, J. Radhakrishnan, and S. Venkatesh proposed a bit-probe scheme based on expanders. Their scheme needs space of $O(n \log m)$ bits, and requires to read only one randomly chosen bit from the memory to answer a query. The answer is correct with probability 2/3 with two-sided errors.

In this paper we show that for the same problem there exists a bit-probe scheme with *one-sided* error that needs space of $O(n \log^2 m + \text{poly}(\log m))$ bits. The difference with the model of Buhrman, Miltersen, Radhakrishnan, and Venkatesh is that we consider a bit-probe scheme with an auxiliary word. This means that in our scheme the memory is split into two parts of different size: the main storage of $O(n \log^2 m)$ bits and a short word of $\log^{O(1)} m$ bits that is pre-computed once for the stored set A and "cached". To answer a query "$x \in A$?" we allow to read the whole cached word and only one bit from the main storage. For some reasonable values of parameters (e.g., for $\text{poly}(\log m) \ll n \ll m$) our space bound is better than what can be achieved by any scheme without cached data (the lower bound $\Omega(\frac{n^2 \log m}{\log n})$ was proven in [11]).

We obtain a slightly weaker result (space of size $n^{1+\delta}\text{poly}(\log m)$ bits and two bit probes for every query) for a scheme that is effectively encodable.

Our construction is based on the idea of naive derandomization, which is of independent interest. First we prove that a random combinatorial object (a graph) has the required properties, and then show that such a graph can be obtained as an outcome of a pseudo-random generator. Thus, a suitable graph can be specified by a short seed of a PRG, and we can put an appropriate value of the seed into the cache memory of the scheme.

1 Introduction

We investigate the static version of the membership problem. The aim is to represent a set $A \subset \{1, \ldots, m\}$ by some data structure so that queries "$x \in A$?" can be

[*] Supported in part by grants RFBR 09-01-00709-a and ANR-08-EMER-008-01.

easily replied. We are interested in cases when the number of elements in the set $n = |A|$ is much less than size m of the universe (e.g., $n = exp\{\text{poly}(\log\log m)\}$ or $n = m^{0.01}$).

In practice, many different data structures are used to represent sets: simple arrays, different variants of height-balanced trees, hash tables, etc. The simplest solution is an array of m bits; to answer a query "$x \in A$?" we need to read a single bit from the memory. However, the size of this data structure is excessive . In a more complicated data structure based on double hashing (Fredman, Komlós, and Szemerédi, [3]) a set is represented as a table of $O(n)$ words of size $\log m$ bits, and a query "$x \in A$?" requires to read $O(1)$ words from the memory. Another popular practical solution is Bloom's filter [1]. This data structure requires only $O(n)$ bits, whatever is the size of the universe; to answer a query we need to read $O(1)$ bits from the memory. The drawback of this method is that we can get false answers to some queries. Only false positives answers are possible (for some $x \notin A$ Bloom's filter answers "yes"), but false negatives are not. For a "typical" set A the fraction of false answers is small.

An interesting approach was suggested by Harry Buhrman, Peter Bro Miltersen, Jaikumar Radhakrishnan, and Venkatesh Srinivasan [11]. They introduced some randomness into the query processing algorithm. That is, the data structure remains static (it is deterministically defined for each set A), but when a query is processed we a toss coins and read randomly chosen bit from the memory. In this model, we allow to return a wrong answer with some small probability. Notice the sharp difference with the Bloom's filter: for *each* x we must correctly reply to the query "$x \in A$?" with probability close to 1.

Buhrman, Miltersen, Radhakrishnan, and Venkatesh investigated both two-sided and one-sided errors. In this paper we will concentrate mostly on one-sided errors: if $x \in A$, then the answer must be always correct, and if $x \notin A$, then some small probability of error is allowed. A trivial information-theoretic bound shows that the size of the structure representing a set A cannot be less that $\log\binom{m}{n} = \Omega(n\log m)$ bits. Surprisingly, this bound can be achieved if we allow two-sided errors and use only single bit probe for each query. This result was proven in [11]. We refer to the scheme proposed their as the BMRV-scheme:

Theorem 1 (two-sided BMRV-scheme, [11]). *For any $\varepsilon > 0$ there is a scheme for storing subsets A of size at most n of a universe of size m using $O(\frac{n}{\varepsilon^2}\log m)$ bits so that any membership query "Is $x \in A$?" can be answered with error probability less than ε by a randomized algorithm which probes the memory at just one location determined by its coin tosses and the query element x.*

The bound achieved in this theorem is very close to the known lower bound. In fact, the trivial lower bound $\log\binom{m}{n}$ can be improved: the less is probability of an error, the greater must be the stored bit vector. For one-sided error schemes a stronger lower bound is known:

Theorem 2 (lower bound, [11]). (a) *For any $\varepsilon > 0$ and $\frac{n}{\varepsilon} < m^{1/3}$, any ε-error randomized scheme which answers queries using one bitprobe must use space $\Omega(\frac{n}{\varepsilon\log 1/\varepsilon}\log m)$.*

(*b*) *Any scheme with* one-sided *error ε that answers queries using at most one bitprobe must use $\Omega(\frac{n^2}{\varepsilon^2 \log(n/\varepsilon)} \log m)$ bits of storage.*

The second part of the theorem above implies that we cannot achieve space of size $O(n \log m)$ with a *one probe* scheme and one-sided error. However we can get very close if we allow $O(1)$ probes (instead of a single probe):

Theorem 3 (one-sided BMRV-scheme, [11]). *Fix any constant $\delta > 0$. There exists a constant t such that the following holds: there is a one-sided $\frac{1}{3}$-error randomized scheme that uses space $O(n^{1+\delta} \log m)$ and answers membership queries with at most t probes.*

The constructions in [11] is not explicit: given the list of elements A, the corresponding scheme is constructed (with some brute force search) in time $2^{\mathrm{poly}(m)}$. Moreover, each membership query requires exponential in m computations.

The crucial element of the constructions in Theorem 1 is an unbalanced expander graph. Existence of a graph with required parameters was proven in [11] probabilistically. We know that such a graph exists and we can find it by brute force search, but we do not know how to construct it explicitly. Since Bassalygo and Pinsker defined expanders [2], many explicit (and poly-time computable) constructions of expander graphs were discovered, see a survey [15]. However, most of known constructions are based on spectral technique that is not suitable to get an expander of degree d with expansion parameter greater than $d/2$, see [8]. This is not enough for the construction used in the proof of Theorem 1 in [11]; we need there a graph with expansion parameter close to d.

There are only very few effective constructions of unbalanced graph with large expansion parameter. One of the known constructions was suggested by Capalbo et al in [12]; its parameters are close to optimal values if the size of the right part of the graph is *constant times* less than the size of the left part of the graph. However, in the BMRV-scheme we need a graph where the right part of the graph is much less than the left part. Some explicit construction suitable for BMRV-scheme was suggested in [14]. The best known explicit construction of a highly unbalanced expander graph was presented in [18]. This beautiful construction is based on the Parvaresh–Vardy code with an efficient list decoding. Thanks to the special structure if this expander, it enjoys some special property of effective decoding. Using this technique, the following variant of Theorem 1 can be proven:

Theorem 4 ([18]). *For any $\delta > 0$ there exists a scheme for storing subset A of size at most n of a universe of size m using $n^{1+\delta} \cdot \mathrm{poly}(\log m)$ bits so that any membership query can be answered with error probability less than ε by a randomized algorithm which probes the memory at one location determined by its coin tosses and the query element x.*

Given the list of elements A, the corresponding storing scheme can be constructed in time $\mathrm{poly}(\log m, n)$. When the storing scheme is constructed, a query for an element x can be calculated in time $\mathrm{poly}(\log m)$.

In Theorems 1, 3, 4, a set A is encoded into a bit string, and when we want to know if $x \in A$, we just read from this string one randomly chosen bit (or $O(1)$

bits in Theorem 3). The obtained information is enough to decide whether x is an element of the set. Let us notice that in all these computations we implicitly use more information. To make a query to the scheme and to process the retrieved bit, we need to know the parameters of the scheme: the size n of the set A, the size m of the universe, and the allowed error probability ε. This auxiliary information is very short (it takes only $\log(m/\varepsilon)$ bits). It does not depend on the stored set A. We assume that it is somehow hardwired into the bitprobe scheme (we say that this information is *cached* in advance by the algorithms that processes queries).

In this paper we consider a more liberal model, where small *cached* information can depend on the set A. Technically, the date stored in our scheme consists of two parts of different size: a small cached string C of length $\text{poly}(\log m)$, and a long bit string B of length $n \cdot \text{poly}\log(m)$. Both these strings are prepared for a given set A of n elements (in the universe of size m). When we need to answer a query "$x \in A$?", we use C to compute probabilistically a position in B and read there one bit. This is enough to answer whether x is an element of A, with a small one-sided error:

Theorem 5. *Fix any constant $\varepsilon > 0$. There exists a one-sided ε-error randomized scheme that includes a string B of length $O(n \log^2 m)$ and an auxiliary word C of length* $\text{poly}(\log m)$*. We can answer membership queries "$x \in A$?" with one bit probe to B. For $x \in A$ the answer is always correct; for each $x \notin A$ probability of error is less than ε.*

The position of the bit probed in A is computed from x and the auxiliary word C in time $\text{poly}(\log m)$.

Remark 1: Schemes with 'cached' auxiliary information that depends on A (not only on its size $n = |A|$ and the size m of the universe) makes sense only if this information is very small. If the size of the cached data is about $\log \binom{m}{n}$ bits, then we can put there the list of all elements of A, so the problem becomes trivial. Since in our construction we need cached information of size $\text{poly}(\log m)$ bits, the result is interesting when $\text{poly}(\log m) \ll n \ll m$, e.g., for $n = exp\{\text{poly}(\log\log n)\}$. Notice that by Theorem 2 the space size $O(n \log^2 m)$ with one-sided error cannot be achieved by any schemes without cached auxiliary information that depends on A.

Remark 2: The model of data structures with cached memory looks useful for practical applications. Indeed, most computer systems contain some hierarchy of memory levels: CPU registers and several levels of processor caches, then random access memory, flash memory, magnetic disks, remote network-accessible drives, etc. Each next level of memory is cheaper but slower. So, it is interesting to investigate the tradeoff between expensive and fast local memory and cheap and slow external memory. There is rich literature on algorithms with *external memory*, see, e.g, surveys [10,16]. Tradeoff between local and external memory is typically studied for *dynamic* data structures. The same time, it is not obvious that fast cache memory of negligible size can help to process queries in a *static* data structure. Since a small cache 'knows' virtually nothing about most objects in the database, so at first sight is seems to be useless. However, Theorem 5 shows that even a very small cache can be surprisingly efficient.

Remark 3: In the proof of Theorem 5 we derandomize a probabilistic proof of existence of some kind of expander graphs. In many works derandomization of probabilistic arguments involves highly sophisticated ad-hoc techniques. But we do derandomization in rather naive and straightforward way: take a value of a suitable pseudo-random number generator and check that with high probability (i.e., for most values of the seed) we obtain the required property. In fact, we observe that several types of generators fit our construction. Since the required property of a graph can be tested in AC^0, we can use the classic Nisan–Wigderson generator or (thanks to the recent result of Braverman [17]) any polylog-independent function. Also the required property of a pseudo-random graph can be tested by a machine with logarithmic space. Hence, we can use Nisan's generator [6]. We stress that we do not need any unproven assumptions to construct all these generators.

Somewhat nonconventional part of our construction is that we consider a 'local' variant of the definition of expanders: we require that the usual expansion property holds not for all sets of vertices but only for one particular set A. This modification makes the property of the expander graph weaker, and this relaxation helps to derandomize the construction.

In Theorem 5 we construct a scheme such that decoding is effective: when the scheme is prepared, we can answer queries "$x \in A$?" in time polynomial in $\log m$. However the encoding (preparing the bits string and the auxiliary word) runs in expected time $\mathrm{poly}(m)$, which is much longer if $m \gg n$. Next theorem claims that the encoding time can be reduced if we slightly increase the space of the scheme:

Theorem 6. *The scheme from theorem 5 can be made effectively encodable in the following sense. Fix any constants $\varepsilon, \delta > 0$. There exists randomized scheme that includes a bit string B of length $n^{1+\delta}\mathrm{poly}(\log m)$ and an auxiliary word C of length $\mathrm{poly}(\log m)$. We can answer membership queries "$x \in A$?" with two bits probe to B. For $x \in A$ the answer is always correct; for $x \notin A$ probability of error is less than ε.*

The position of the bit probed in A is computed by x and the auxiliary word C in time $\mathrm{poly}\log(m)$. Given A, the entire scheme (the string B and the word C) can be computed probabilistically in average time $\mathrm{poly}(n, \log m)$.

The rest of the paper is organized as follows. In Section 2 we remind the main ideas in the BMRV-scheme. We prove Theorem 5 in Section 3, and Theorem 6 in Section 4. In Conclusion we discuss some open questions.

2 How BMRV-Scheme Works

Let us explain the main ideas of the proof of Theorem 1 in [11]. The construction is based on highly unbalanced *expanders.*

Definition 1. *A bipartite graph $G = (L, R, E)$ (with left part L, right part R and set of edges E) is called (m, s, d, k, δ)-expander if L consists of m vertices, R consists of s vertices, degree of each vertex in L is equal to d, and for each subset of vertices $A \subset L$ of size at most k the number of neighbors is at least $(1 - \delta)d|A|$.*

We use a standard notation: for a vertex v we denote by $\Gamma(v)$ the set of its neighbors; for a set of vertices A we denote by $\Gamma(A)$ the set of neighbors of A, i.e., $\Gamma(A) = \cup_{v \in A}\Gamma(v)$. So, the definition of expanders claims that for all small enough sets A of vertices in the left part of the graph, $|\Gamma(A)| \geq (1 - \delta)d|A|$ (the maximal size of $|\Gamma(A)|$ is obviously $d|A|$, since degree of all vertices on the left is equal to d). The argument below is based on the following combinatorial property of an expander:

Lemma 1 (see [12]). *Let ε be a positive number, and G be an (m, s, d, k, δ)-expander with $\delta \leq \varepsilon/4$. Then for every subset $A \subset L$ such that $|A| \leq k/2$, the number of vertices $x \in L \setminus A$ such that*

$$|\Gamma(x) \cap \Gamma(A)| \geq \varepsilon d$$

is not greater than $|A|/2$.

Let G be a (m, s, d, k, δ)-expander with $\delta < \varepsilon/4$. The storage scheme is defined as follows. We identify a set $A \subset \{1, \dots, m\}$ of size n ($n \leq k/2$) with a subset of vertices in the left part of the graph. We will represent it by some labeling (by ones and zeros) on the vertices of the right part of the graph. We do it in such a way that the vast majority (at least $(1 - \varepsilon)d$) of neighbors of each vertex v from the left part of the graph correctly indicate whether $v \in A$. More precisely, if $v \in A$ then at least $(1 - \varepsilon)d$ of its neighbors in R are labeled by 1; if $v \in L \setminus A$ then at least $(1 - \varepsilon)d$ of its neighbors in R are labeled by 0. Thus, querying a random neighbor of v will return the right answer with probability $> 1 - \varepsilon$.

It remains to explain why such a labeling exists. In fact, it can be constructed by a simple greedy algorithm. First, we label all neighbors of A by 1, and the other vertices on the left by 0. This labeling classifies correctly all vertices in A. But it can misclassify some vertices outside A: some vertices in $L \setminus A$ can have too many (more than εd) neighbors labeled by 1. Denote by B the set of all these "erroneous" vertices. We relabel all their neighbors, i.e., all vertices in $\Gamma(B)$ to 0. This fixes the problem with vertices outside A, but it can create problems with some vertices in A. We take the set of all vertices in A that became erroneous (i.e., vertices in A that have at least εd neighbors in $\Gamma(B)$), and denote this set of vertices by A'. Then we relabel all $\Gamma(A')$ to 1. This operation create new problems in some set of vertices $B' \subset B$, relabel $\Gamma(B')$ to 0, etc. In this iterative procedure we get a sequence of sets $A \supset A' \supset A'' \supset \dots$ whose neighbors are relabeled to 1 on steps $1, 3, 5, \dots$ of the algorithm, and $B \supset B' \supset B'' \supset \dots$ whose neighbors are relabeled to 0 on iterations $2, 4, 6, \dots$ respectively. Lemma 1 guarantees that the number of the erroneous vertices on each iteration reduces by a factor of 2 ($|B| \leq |A|/2$, $|A'| \leq |B|/2$, etc.). Hence, in $\log m$ steps the procedure terminates.

To organize a storing scheme (and to estimate its size) for a set A of size n in the universe of size m, we should construct an $(m, s, d, k = 2n, \delta = \varepsilon/4)$-expander. Parameters m, k, δ of the graph are determined directly by the parameters of the desired scheme (by the size of A and the universe and the allowed error probability ε). We want to minimize the size of the left part of the graph s, which is the size of the stored data. Existence of expanders with good parameters can be proven by probabilistic arguments:

Lemma 2 ([11]). *For all integers m, n and real $\varepsilon > 0$ there exists an $(m, s = O(\frac{n \log m}{\varepsilon^2}), d = \frac{\log m}{\varepsilon}, n, \varepsilon)$-expander. Moreover, the vast majority of bipartite graphs with n vertices on the left, $s = \frac{100 n \log m}{\varepsilon^2}$ vertices on the right, and degree $d = \frac{\log m}{\varepsilon}$ at all vertices on the left are expanders.*

Given the parameters m, n, ε, we can find an $(m, O(\frac{n \log m}{\varepsilon^2}), \frac{\log m}{\varepsilon}, n, \varepsilon)$-expander by brute force search. This can be done by a deterministic algorithm in time $2^{\mathrm{poly}(m/\varepsilon)}$. Hence, to construct the bit-probe structure defined above we need exponential time. Moreover, when the structure is constructed and we want to answer a query "$x \in A$?" we need to read only one bit from the stored bit string. But to select the position of this bit we need again to reconstruct the expander graph, which requires exponential computations. We can keep the structure of the computed graph in "cache" (compute the graph once, and then re-use it every time a new query should be answered). But then the size of this "cached data" (the size of the graph) becomes much greater than m, which makes the bit-probe scheme useless (it is cheaper to store A as a trivial m-bits array).

In [18] a nice and very powerful explicit construction of expanders was suggested:

Theorem 7 ([18]). *Fix an $\varepsilon > 0$ and $\delta > 0$. For all integers m, n there exists an explicit $(m, s = n^{1+\delta} \cdot \mathrm{poly}(\log m), d = \mathrm{poly}(\log m), n, \varepsilon)$-expander such that for an index of a vertex v from the left part (a binary representation of an integer between 1 and m) and an index of an outgoing edge (a binary representation of an integer between 1 and d), the corresponding neighbor on the right part of the graph (an integer between 1 and s) can be computed in time polynomial in $\log m$.*

Also, the following effective decoding algorithm exists. Given a set of vertices T from the right part of the graph, we can compute the list of vertices in the left part of the graph that have at least $(4\varepsilon d)$ neighbors in T, i.e.,

$$S = \{v \ : \ |\Gamma(v) \cap T| \geq 4\varepsilon d\},$$

in time $\mathrm{poly}(|S|, n, \log m)$.

Theorem 4 is proven by plugging the expander from Theorem 7 in the general scheme explained above, see details in [18].

3 Proof of Theorem 5

3.1 Refinement of the Property of ε-Reduction

The construction of bit-probe scheme for a set A of size n in the m-elements universe (with probability of an error bounded by some ε) explained in the previous section involves an (m, s, d, k, δ)-expander with $s = O(\frac{n}{\varepsilon^2} \log m)$ and $d = O(\frac{1}{\varepsilon} \log m)$. Such a graph contains dm edges (degree of each vertex on the left is d). The list of all its edges can be specified by a string of $dm \log s$ bits: we sort all edges by their left ends, and specify for each edge its right end. Denote the size of the description of this graph by $N = dm \log s$.

In what follows we will assume that number s is a power of 2 (this will increase the parameters of the graph by only a factor at most 2). So, we may assume that every string of $N(m, s, d) = dm \log s$ bits specifies a bipartite graph with m vertices on the left, s vertices on the right and degree d on the left. Lemma 2 claims that most of these bits string of length N describe an (m, s, d, k, δ)-expander. By Lemma 1, if a graph is an expander with these parameters, then for $\varepsilon = 4\delta$, for every set $A \subset L$ of size less than $k/2$ the following *reduction property* holds:

Combinatorial Property 1 (ε-reduction property). *For every subset $A \subset L$ such that $|A| \le k/2$, the number of vertices $x \in L \setminus A$ such that*

$$|\Gamma(x) \cap \Gamma(A)| \ge \varepsilon d$$

is not greater than $|A|/2$.

This property was the main ingredient of the BMRV-scheme. In our bit-probe scheme we will need another variant of Property 1:

Combinatorial Property 2 (strong ε-reduction). *Let $G = (L, R, E)$ be a bipartite graph, and $A \subset L$ be a subset of vertices from the left part. We say that the* strong ε-reduction property *holds for A in this graph if for all $x \in L \setminus A$*

$$|\Gamma(x) \cap \Gamma(A)| \le \varepsilon d.$$

Lemma 3. *Fix an $\varepsilon > 0$. For all integers m, n, for every $A \subset \{1, \dots, m\}$ of size n there exists a bi-partite graph $G = (L, R, E)$ such that*

- *$|L| = m$ (the size of the left part);*
- *$|R| = 2d^2 n = O(n \log^2 m)$ (the size of the right part);*
- *degree of each vertex in the left part is $d = \frac{2 \log m}{\varepsilon} = O(\log m)$;*
- *the property of strong ε-reduction holds for the set A (we identify it with a subset of vertices in left part of the graph).*

Moreover, the property of strong ε-reduction for A holds for the majority of graphs with the parameters specified above.

The order of quantifiers is important here: we do not claim that in a random graph the strong ε-reduction property holds for all A; we say only that for every A the strong ε-reduction is true in a random graph.

Proof of lemma: Let v be any vertex in $L \setminus A$. We estimate probability that at least εd neighbors of x are at the same time neighbors of A (assuming that all edges are chosen at random independently). There are $\binom{d}{\varepsilon d}$ choices of εd vertices among all neighbors of v. Hence,

$$\text{Prob}[|\Gamma(v) \cap \Gamma(A)| \ge \varepsilon d] \le \binom{d}{\varepsilon d} \cdot \left(\frac{|\Gamma(A)|}{|R|}\right)^{\varepsilon d} \le d^{\varepsilon d} \cdot \left(\frac{dn}{2d^2 n}\right)^{\varepsilon d} = \left(\frac{1}{2}\right)^{2 \log m}$$

This probability is less than $1/m^2$ (for each vertex v). So, the expected number of vertices $v \in L$ such that $|\Gamma(v) \cap \Gamma(A)| \geq \varepsilon d$, is less than $1/m < 1/2$. Hence, the strong ε-reduction property holds for A for more than 50% of graphs.

3.2 Testing the Property of Strong ε-Reduction

Lemma 3 implies that a graph with the strong reduction property for A exists. Given A, we can find such a graph by brute force search. But we cannot use such a graph in our bit-probe scheme even if we do not care about computation complexity: the choice of the graph depends on A, and the size of the graph is too large to include it into the scheme. We need to find a suitable graph with a short description. We will do it using pseudo-random generators ('random' graphs will be parameterized by the seed of a generator).

Property 2 is a property of a graph and a set of vertices A in graph. We can interpreted it as a property of an N-bits string (that determines a graph) and some $A \subset \{1, \ldots, m\}$. Lemma 3 claims that for every A, for a randomly chosen graph (a randomly chosen N-bits string) with high probability the strong reduction property is true. We want to show that the same is true for a *pseudo-random* graph. At first, we observe that the strong reduction property can be tested by an AC^0 circuit (a Boolean circuit of bounded depth, with polynomial number of gates *and*, *or* with unbounded fan-in, and *negations*).

Indeed, we need to check for each vertex $v \in L \setminus A$ that the number of vertices in $\Gamma(v) \cap \Gamma(A)$ is not large. For each vertex w in the right part of the graph we can compute by an AC^0-circuit whether $w \in \Gamma(A)$. For each v from the left part of the graph and each w from the right part of the graph we check by an AC^0-circuit whether there is an edge (v, w) in the graph. Thus, it remains to count for each $v \in L \setminus A$ the number of $w \in R$ such that $w \in \Gamma(A)$ and there exists an edge between v and w.

In AC^0 we cannot compute threshold functions with linear number of inputs (e.g., the majority function is not in AC^0, [4]). However, we can compute thresholds for *logarithmic* number of arguments. Thresholds $Th_d^{\varepsilon d}$ for $d = O(\log N)$ can be represented by a CNF of size $2^{O(d)} = \mathrm{poly}(N)$. This is exactly what we need: we want to check for each $v \in L \setminus A$ that the number of vertices in $\Gamma(v) \cap \Gamma(A)$ is not greater than εd, and $d = O(\log m)$. We combine together these circuits for all v and get an AC^0-circuit that tests the property of strong ε-reduction.

3.3 Pseudo-random Graphs

We need to generate a pseudo-random string of N bits that satisfies the strong ε-reduction property (for some fixed set A). We know that (i) by Lemma 3, for a uniformly distributed random string this property is true with high probability, and (ii) this property can be checked in AC^0. It remains to choose a pseudo random generator that fools this particular AC^0-circuit. There exist several generators that fools such distinguishers. Below we mention the known solutions.

The first solution: the generator of Nisan–Wigderson. The classic way to fool an AC^0 circuit is the Nisan–Wigderson generator:

Theorem 8 (Nisan–Wigderson generator, [7]). *For every constant c there exists an explicit family of functions*

$$G_m : \{0,1\}^{\mathrm{poly}(\log N)} \to \{0,1\}^N$$

such that for for any family of circuits C_N (with N inputs) of polynomial in N size and depth c, the difference between $\mathrm{Prob}[C_N(y) = 1]$ *and* $\mathrm{Prob}[C_N(G_m(x))]$ *tends to zero (faster than* $1/\mathrm{poly}(N)$*).*

The generator is effective: generator's value $G_m(x)$ can be computed from a given x in time $\mathrm{poly}(\log N)$*.*

From this theorem and Lemma 3 it follows that for each $A \subset \{1, \ldots, m\}$ of size at most n, for most values of the seed of the Nisan–Wigderson generator G_m, a pseudo-random graph $G_m(x)$ satisfies the strong ε-expansion property for A.

The second solution: polylog-independent strings. M. Braverman proves that all polylog-independent functions fool AC^0-circuits:

Theorem 9 (Braverman, [17]). *Let $\mathcal{C}$ be a Boolean circuit of depth r and size M, ε be a positive number, and*

$$D = \left(\log \frac{M}{\varepsilon}\right)^{\kappa r^2}$$

(for some absolute constant κ). Then $\mathcal{C}$ cannot distinguish between the uniform distribution U and any D-independent distribution μ on its inputs:

$$|\mathrm{Prob}_\mu[\mathcal{C} \textit{ accepts}] - \mathrm{Prob}_U[\mathcal{C} \textit{ accepts}]| < \varepsilon$$

It follows that instead of the Nisan–Wigderson generator we can take any $(\log^c n)$-independent function (for large enough constant c). The standard solution is a polynomial of degree $r = \log^c n$ over a finite field of size about N (seeds of this 'pseudo-random generator' are coefficients of a polynomial). Alternatively, other constructions of polylog-independent functions can be used. E.g., the construction from [5,9] provides a family of $(\log^c n)$-independent functions with very fast evaluation algorithm, and each function is specified by $\mathrm{poly}(\log n)$ bits (so, the size of the *seed* is again poly-logarithmic).

The third solution: the generator of Nisan. The property of strong ε-reduction can be tested by a Turing machine with logarithmic working space. Technically we need a machine with

– *advice tape:* read-only, two-way tape, where the list of elements of A is written;
– *input tape:* read-only tape with random (or pseudo-random) bits, with logarithmic number of passes (the machine is allowed to pass along the input on this tape only $O(\log N)$ times);
– *index tape:* read-only, two-way tape with logarithmic additional information;
– *work tape* that is two-way and read-write; the zone of the working tape is restricted to $O(\log N)$.

We interpret the content of the input tape as a list of edges of a random (or pseudo-random) graph $G = (L, R)$. The content of the index tape is understood as an index of a vertex $v \in L \setminus A$. The machine reads the bits from the 'input tape' (understood as a list of edges of a random graph) and checks that most neighbors of v does not belong to the set of neighbors of A. The machine needs to read the input tape $2d = O(\log N)$ times (where d is degree of v): on the first pass we find the index of the first neighbor of v; on the second pass we check whether this neighbor of v is incident to any vertex of A; then we find the second neighbor of v, check whether it is is incident to any vertex of A, etc. The machine *accepts* the input if $|\Gamma(v) \cap \Gamma(A)| < \varepsilon d$.

We can use Nisan's generator [6] to fool this machine. Indeed, this machine and the tested property fits the general framework of [13], where Nisans generator was used to derandomize several combinatorial constructions. The only nonconventional feature in our argument is that the input tape is not *one-way*: we are allowed to read the random bits not once but logarithmic number of times. However, we still can apply Nisan's technique. David, Papakonstantinou, and Sidiropoulos recently observed (see [20]) that a log-space machine with logarithmic (and even poly-logarithmic) number of passes on the input tape is fooled by Nisan's generator with the seed size $\mathrm{poly}(\log N)$.

Now we are ready to prove Theorem 5. We fix an $\varepsilon > 0$ and a set $A \subset \{1, \dots, m\}$ of size m. Let G_m be one of the pseudo-random generators discussed above. For all these generators, for most values of the seed z the values $G_m(z)$ encodes a graph such that the strong ε-reduction property holds for A. Let us fix one of such seeds. We label by 1 all vertices in $\Gamma(A)$ and by 0 all other vertices in R in the graph encoded by the string $G_m(z)$.

The seed value z makes the "auxiliary word" C, and the specified above labeling of the right part of the graph makes the bits string B. To answer a query "$x \in A$?" we take a random neighbor of x in the graph and check its label. If the label is 1, we answer "$x \in A$"; otherwise we answer $x \notin A$.

If $x \in A$ then there are no errors, since all neighbors of A are labeled by 1. If $x \notin A$ then probability of an error is bounded by ε because of the strong ε-reduction property. We can answer a query in polynomial (in $\log m$) time since the generators under consideration are effectively computable.

3.4 Complexity of Encoding

The disadvantage of this construction is non-effective encoding procedure. We know that for most seeds z the corresponding graph $G_m(z)$ enjoys the strong ε-reduction property. However, we need the brute force search over all vertices $v \in L \setminus A$ (polynomial in m but not in $\log m$) to check this property for any particular seed. Thus, we have a probabilistic encoding procedure that runs in *expected* time $\mathrm{poly}(m)$: we choose random seeds until we find one suitable for the given A.

In the next section we explain how to make the encoding procedure more effective (in expected time $\mathrm{poly}(n, \log m)$) for the following price: we will need a slightly greater size of the data storage, and we will take 2 bit probes instead of one at each query.

4 Proof of Theorem 6: Effective Encoding

To obtain a scheme with effective encoding and decoding we combine two constructions: the explicit expander from [18] and a pseudo-random graph from the previous section.

We fix an n-element set A in the universe $\{1, \ldots, m\}$. Now we construct two bipartite graphs that share the same left part $L = \{1, \ldots, m\}$. The first graph is the explicit $(m, s = n^{1+\delta} \cdot \mathrm{poly}(\log m), d = \mathrm{poly}(\log m), n, \varepsilon)$-expander $G_1 = (L, R_1, E_1)$ from [18] with an effective decoding algorithm. We do the first two steps from the encoding procedure of the BMRV-scheme explained in Section 2. At first we label all vertices in $\Gamma(A)$ by 1 and other vertices by 0. Denote the corresponding labeling (which is a $n^{1+\delta} \cdot \mathrm{poly}(\log m)$-bits string) by C_1. Then we find the list of vertices outside A that have too many 1-labeled neighbors:

$$B := \{v \in L \setminus A \ : \ |\Gamma(x) \cap \Gamma(A)| \geq \varepsilon d\}.$$

We do not re-label neighbors of B, but we will use this set later (to find B effectively, we need the property of effective decoding of the graph).

Let $v \in L$ be a vertex in the left part of the graph. There are three different cases:

- if v belongs to A then all neighbors of v are labeled by 1;
- if v does not belong to $A \cup B$, then a random neighbor of x with probability $> (1 - \varepsilon)d$ is labeled by 0;
- if v belongs to B, we cannot say anything about the vast majority of its neighbors.

Thus, if we take a random neighbor of v and see label 0 in C_1 then we can say that this point does not belong to A. If we see label 1 then more detailed investigation is needed. This investigation will involve the second part C_2 of the scheme defined below.

Now our goal is to distinguish between A and B. To this end we take a pseudo-random graph $G_2 = (L, R_2, E_2)$ specified by a value of a pseudo-random generator $G_m(z)$ (any one from the previous section). We need a *restricted on B version of the strong ε-reduction property*:

For every $v \in B$, at most εd vertices in $\Gamma(v)$ belong to $\Gamma(A)$.

Set B is of size at most $|A|/2$ (Lemma 1), and it can be effectively computed from A (effective decoding property of the graph). Hence, for a given z we can check the property above in time $\mathrm{poly}(n, \log m)$. We know that for the majority of seeds z the graph $G_m(z)$ satisfies the strong ε-reduction property, i.e., all vertices outside A have at most εd neighbors in $\Gamma(A)$. We cannot effectively check this general property (we cannot check it for *all* vertices in the universe), but we can check its restricted version (i.e., only for vertices in B).

Thus, in average time $\mathrm{poly}(n, \log m)$ we can probabilistically find some seed x such that the restricted (on B) version of the strong ε-reduction property is true. In the corresponding graph we denote by 1 all vertices in $\Gamma(A)$, and by 0 all vertices of the right part of the graph outside $\Gamma(A)$. We denote this labeling (a $O(m \log^2 n)$-bits string) by C_2 and take it as the second part of the data storage. The seed value x is taken as 'cached' memory.

The decoding procedure works as follows. Given $x \in \{1, \ldots, m\}$, we take its random neighbor in both constructed graphs and look at their labels (bits from C_1 and C_2 respectively).

– if the first label is 0, we say that $x \notin S$;
– if the first label is 1 and the second bit is 0 then we say that $x \notin S$.
– if both labels are is 1 then we say that $x \in S$.

If $x \notin A \cup B$ then by definition of B we get that the procedure above with probability $> (1 - \varepsilon)$ returns the correct answer. If $x \in A$ then by construction both labels are equal to 1. If $x \in B$, then we have no guarantee about labels in C_1; but from the restricted strong reducibility property it follows that with probability $> (1 - \varepsilon)$ the second label is 0. Thus, we have one-sided error probability bounded by ε.

5 Conclusion

In this paper we constructed an effective probabilistic bit-probe scheme with one-sided error. The used space is close to the trivial information-theoretic lower bound $\Omega(n \log m)$. The scheme answers queries '$x \in A$?' with a small one-sided error and requires only poly-logarithmic (in the size of the universe) cached memory and *one bit* (*two bits* in the version with effective encoding) from the main part of the memory.

For reasonable values of parameters (for $n \gg \mathrm{poly}(\log m)$) the size of our scheme $O(n \log^2 m)$ with a cache of size $\mathrm{poly}(\log m)$ is below the lower bound $\Omega(n^2 \log m)$ proven in [11] for one-probe schemes with one-sided errors without cached data dependent on A. The gap between our upper bounds and the trivial lower bound is a $(\log m)$-factor.

The following questions remain open: How to construct a bit-probe memory scheme with one-sided error and effective encoding and decoding that requires to read only *one* bit from the main part of memory to answer queries? What is the minimal size of the cached memory required for a bit-probe scheme with one-sided error, with space of size $O(n \log m)$?

The author thanks Daniil Musatov for useful discussions, and anonymous referees for deep and very helpful comments.

References

1. Bloom, B.: Space-time trade-offs in hash coding with allowable errors. Communications of ACM 13(7), 422–426 (1970)
2. Bassalygo, L.A., Pinsker, M.S.: The complexity of an optimal non-blocking commutation scheme without reorganization. Problems of Information Transmission 9, 64–66 (1974)
3. Fredman, M.L., Komlós, J., Szemerédi, E.: Storing a sparse table with $O(1)$ worst case access time. Journal of the Association for Computing Machinery 31(3), 538–544 (1984)
4. Håstad, J.: Almost optimal lower bounds for small depth circuits. In: Proc. of 18th ACM STOC, pp. 6–20 (1986)
5. Siegel, A.: On universal classes of fast high performance hash functions, their time-space trade-off, and their applications. In: Proc. of 30th IEEE FOCS, pp. 20–25 (1989)
6. Nisan, N.: Pseudorandom generators for space-bounded computation. Combinatorica 12(4), 449–461 (1990); Preliminary version: STOC 1990
7. Nisan, N., Wigderson, A.: Hardness vs Randomness. J. Comput. Syst. Sci. 49(2), 149–167 (1994)
8. Kahale, N.: Eigenvalues and expansion of regular graphs. Journal of the ACM 42(5), 1091–1106 (1995)
9. Siegel, A.: On universal classes of extremely random constant time hash functions and their time-space tradeoff. Technical Report TR1995-684, Courant Institute, New York University (April 1995)
10. Vitter, J.S.: External memory algorithms and data structures. ACM Comput. Surv. 33(2), 209–271 (2001)
11. Buhrman, H., Miltersen, P.B., Radhakrishnan, J., Srinivasan, V.: Are bitvectors optimal? Siam J. on Computing 31(6), 1723–1744 (2002)
12. Capalbo, M.R., Reingold, O., Vadhan, S.P., Wigderson, A.: Randomness Conductors and Constant-Degree Lossless Expanders. In: Proc. of the 34th ACM STOC, pp. 659–668
13. Sivakumar, D.: Algorithmic derandomization via complexity theory. In: Proc. ACM STOC 2002, pp. 619–626 (2002)
14. Ta-Schma, A.: Storing information with extractors. Information Processing Letters 83, 267–274 (2002)
15. Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bulletin of the American Mathematical Society 43(4), 439–561 (2006)
16. Vitter, J.S.: Algorithms and Data Structures for External Memory. Series on Foundations and Trends in Theoretical Computer Science. Now Publishers, Hanover (2008)
17. Braverman, M.: Poly-logarithmic Independence Fools AC^0 Circuits. In: IEEE Conference on Computational Complexity 2009, pp. 3–8 (2009)
18. Guruswami, V., Umans, C., Vadhan, S.: Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. Journal of the ACM 56(4) (2009)
19. Musatov, D.: Theorems about space-bounded Kolmogorov complexity obtained by "naive" derandomization. In: Proc. Computer Science in Russia (2011); Preliminary version: arXiv:1009.5108 (2010)
20. David, M., Papakonstantinou, P.A., Sidiropoulos, A.: How strong is Nisan's pseudorandom generator? (2010) Electronic preprint, `http://itcs.tsinghua.edu.cn/~papakons/pdfs/nisan_passes.pdf`

Improving the Space-Bounded Version of Muchnik's Conditional Complexity Theorem via "Naive" Derandomization[⋆]

Daniil Musatov

Lomonosov Moscow State University
musatych@gmail.com

Abstract. Many theorems about Kolmogorov complexity rely on existence of combinatorial objects with specific properties. Usually the probabilistic method gives such objects with better parameters than explicit constructions do. But the probabilistic method does not give "effective" variants of such theorems, i.e. variants for resource-bounded Kolmogorov complexity. We show that a "naive derandomization" approach of replacing these objects by the output of Nisan-Wigderson pseudo-random generator may give polynomial-space variants of such theorems.

Specifically, we improve the preceding polynomial-space analogue of Muchnik's conditional complexity theorem. I.e., for all a and b there exists a program p of least possible length that transforms a to b and is simple conditional on b. Here all programs work in polynomial space and all complexities are measured with logarithmic accuracy instead of polylogarithmic one in the previous work.

1 Introduction

Many statements about Kolmogorov complexity may be proven by applying some combinatorial constructions like expanders or extractors. Usually these objects are characterized by some parameters, and one may say which parameters are "better". Very often the probabilistic method allows us to obtain these objects with much better parameters than explicit constructions do. But exploiting the probabilistic method causes exponential-space brute-force search of an object satisfying the necessary property. And if this search is performed while describing some string to obtain an upper bound on its complexity then this bound cannot be repeated for polynomial-space complexity. On the other hand, replacing the probabilistic method by an explicit construction weakens the statement due to worse parameters.

We present a technique that combines advantages of both probabilistic and explicit construction methods. The key idea is to substitute a random object with a pseudo-random one obtained by the Nisan-Wigderson pseudo-random generator and still possessing the necessary property. We employ indistinguishability

[⋆] Supported by ANR Sycomore, NAFIT ANR-08-EMER-008-01 and RFBR 09-01-00709-a grants.

of Nisan-Wigderson generator's output from a random string by boolean circuits of constant depth and polynomial size. If the necessary property can be tested by such circuits then it holds for a pseudo-random object as well as for a truly random object. Unfortunately, it is not clear how to build such a circuit for the extractor property and similar ones, so we relax the property in a way that allows both proving the theorem and building polynomial constant-depth circuits.

This "naive derandomization" idea has been recently applied by Andrei Romashchenko ([18]) in another situation: he constructs a probabilistic bit-probe scheme with one-sided error for the membership problem.

By exploiting the new method we improve previous result [12] generalizing Muchnik's conditional complexity theorem. The original theorem [9] states that for all a and b of length n there exists a program p of length $\mathrm{C}(a|b) + O(\log n)$ that transforms b to a and has complexity $O(\log n)$ conditional on a. In [12] this result is restated for space-bounded complexity with gap rised from $O(\log n)$ to $O(\log^3 n)$. The main idea was to employ a property of extractors, proven in [2]: in an extractor graph in every sufficiently big subset S of left part there are few vertices with all right-part neighbours having indegree from S twice greater than average. We refer to it as to "low-congesting property". An explicit extractor construction yields a space-bounded version of the theorem with polylogarithmic precision. In this paper we replace an explicit extractor by a pseudo-random graph, that does not necessary have the extractor property, but enjoys the low-congesting property for "relevant" subsets S. This replacement leads to decreasing the precision back to logarithmic term.

The rest of the paper is organized as follows. In Sect. 2 we give formal definitions of all involved objects and formulate necessary results. In Sect. 3 we formally state our space-bounded variant of Muchnik's theorem and specify all details of the proof.

2 Preliminaries

2.1 Kolmogorov Complexity

Let $\mathcal{V}$ be a two-argument Turing machine. We refer to the first argument as to the "program" and to the second argument as to the "argument". (Plain) Kolmogorov complexity of a string x conditional on y with respect to $\mathcal{V}$ is the length of a minimal program p that transforms y to x, i.e.

$$\mathrm{C}_{\mathcal{V}}(x \mid y) = \min\{p \colon \mathcal{V}(p, y) = x\}$$

There exists an optimal machine $\mathcal{U}$ that gives the least complexity up to an additive term. Specifically, $\forall \mathcal{V}\, \exists c\, \forall x, y\, \mathrm{C}_{\mathcal{U}}(x|y) < \mathrm{C}_{\mathcal{V}}(x|y) + c$. We employ such a machine $\mathcal{U}$, drop the subscript and formulate all theorems up to a constant additive term. The unconditional complexity $\mathrm{C}(x)$ is the complexity with empty condition $\mathrm{C}(x \mid \varepsilon)$, or the length of a shortest program *producing* x.

Now we define the notion of resource-bounded Kolmogorov complexity. Roughly speaking, it is the length of a minimal program that transforms y to x efficiently. Formally, Kolmogorov complexity of a string x conditional on y in time

t and space s with respect to $\mathcal{V}$ is the length of a shortest program p such that $\mathcal{V}(p, y) = x$, and $\mathcal{V}(p, y)$ works in t steps and uses s cells of memory. This complexity is denoted by $\mathrm{C}_{\mathcal{V}}^{t,s}(x \mid y)$. Here the choice of $\mathcal{V}$ alters not only complexity, but also time and space bounds. However, there still exists an optimal machine in the following sense:

Theorem 1. *There exists a machine $\mathcal{U}$ such that for any machine $\mathcal{V}$ there exists a constant c such that for all x, y, s and t it is true that $\mathrm{C}_{\mathcal{U}}^{s,t}(x \mid y) \leq \mathrm{C}_{\mathcal{V}}^{cs,ct\log t}(x \mid y) + c$.*

In our paper we deal only with space bounds, so we drop the time-bound superscript in all notations. Also, we drop the machine subscript again and formulate all theorems with a constant additive term in all complexities and a constant multiplicative term in all space bounds.

2.2 Extractors

An extractor is a function that extracts randomness from weak random sources. A k-weak random source of length n is a probabilistic distribution on $\{0,1\}^n$ with minentropy greater than k, that is, no particular string occurs with probability greater than 2^{-k}. An extractor with parameters n, m, d, k, ε is a function $\mathrm{Ext}\colon \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$, such that for any independent k-weak random source x of length n and uniform distribution u on $\{0,1\}^d$ the induced distribution $\mathrm{Ext}(x, u)$ on $\{0,1\}^m$ is ε-close to uniform, that is, for any set $Y \subset \{0,1\}^m$ its probability differs from its fraction by at most ε. This is interpreted as follows: an extractor receives n weakly random bits and d truly random bits independent from the first argument and outputs m almost random bits.

Like any two-argument function, an extractor may be viewed as a bipartite (multi-)graph: the first argument indexes a vertex in the left part, the second argument indexes an edge going from this vertex, and the value indexes a vertex in the right part which this edge directs to. That is, the graph has $N = 2^n$ vertices on the left, $M = 2^m$ vertices on the right, and all left-part vertices have degree $D = 2^d$. Throughout the paper, we say that a bipartite graph has parameters (n, m, d) if the same holds for it. For the sake of clarity we usually omit these parameters in extractor specifications. The extractor property may be also formulated in terms of graphs: for any left-part subset S of size greater than $K = 2^k$ and for any right-part subset Y the fraction of edges directing from S to Y among all edges directing from S differs from the fraction of Y among all right-part vertices by at most ε. Put formally, $||Y|/M - E(S, Y)/E(S, M)| < \varepsilon$, where $E(S, T)$ is the number of edges diredting from S to T and M slightly abusively denotes both the right part and the number of vertices in it. A proof of equivalence may be found in [16].

It is proven by the probabilistic method (see, for example, [16]) that for all n, k and ε there exists an extractor with parameters $d = \log(n-k) + 2\log(1/\varepsilon) + O(1)$ and $m = k + d - 2\log(1/\varepsilon) - O(1)$. Nevertheless, no explicit (that is, running in polynomial time) construction of such an extractor is known. Best current results ([17], [20]) for $m = k$ use $d = \operatorname{polylog} n$ truly random bits. A number of explicit extractors including those in [7], [4] and [5] use $O(\log n)$ truly random bits but output only $(1-\alpha)k$ almost random bits. Such constructions are insufficient for our goals.

2.3 Nisan-Wigderson Generators

The Nisan-Wigderson pseudo-random generator is a deterministic polynomal-time function that generates n pseudorandom bits from polylog(n) truly random bits. The output of such a generator cannot be distinguished from a truly random string by small circuits. Specifically, we exploit the following theorem from [15]. (The statement was initially proven by Nisan in paper [14]).

Theorem 2. *For any constant d there exists a family of functions $G_n\colon \{0,1\}^k \to \{0,1\}^n$, where $k = O(\log^{2d+6} n)$, such that two properties hold:*

Computability: *G is computable in workspace* poly(k)*;*
Indistinguishability: *For any family of circuits C_n of size* poly(n) *and depth d for any positive polynomial p for all large enough n it holds that:*

$$|\mathrm{Prob}_x\{C_n(G_n(x)) = 1\} - \mathrm{Prob}_y\{C_n(y) = 1\}| < \frac{1}{p(n)},$$

where x is distributed uniformly in $\{0,1\}^k$ and y — in $\{0,1\}^n$.

By rescaling the parameters we get the following

Corollary 1. *There exists a family of functions $G_n\colon \{0,1\}^k \to \{0,1\}^N$, where $k =$ poly(n) and $N = 2^{O(n)}$, such that two properties hold:*

- *G is computable in workspace* poly(n)*;*
- *For any family of circuits C_n of size $2^{O(n)}$ and constant depth, for any constant c and for all large enough n it holds that:*

$$|\mathrm{Prob}_x\{C_n(G_n(x)) = 1\} - \mathrm{Prob}_y\{C_n(y) = 1\}| < 2^{-cn}.$$

The last corollary implies the following basic principle:

Lemma 1. *Let $\mathcal{C}_n$ be some set of combinatorial objects encoded by boolean strings of length $2^{O(n)}$. Let $\mathcal{P}$ be some property satisfied for fraction at least α of objects in $\mathcal{C}_n$ that can be tested by a family of circuits of size $2^{O(n)}$ and constant depth. Then for sufficiently large n the property $\mathcal{P}$ is satisfied for a fraction at least $\alpha/2$ of values of G_n, where G_n is the function from the previous corollary.*

2.4 Constant-Depth Circuits for Approximate Counting

It is well-known that constant-depth circuits cannot compute the majority function. Moreover, they cannot compute a general threshold function that equals 1 if and only if the fraction of 1's in its input exceeds some threshold α. Nevertheless, one can build such circuits that compute threshold functions approximately. Namely, the following theorem holds:

Theorem 3 ([1]). *Let $\alpha \in (0,1)$. Then for any (constant) ε there exists a constant-depth and polynomial-size circuit C such that $C(x) = 0$ if the fraction of 1's in x is less than $\alpha - \varepsilon$ and $C(x) = 1$ if the fraction of 1's in x is greater than $\alpha + \varepsilon$.*

Note that nothing is promised if the fraction of 1's is between $\alpha - \varepsilon$ and $\alpha + \varepsilon$. So, the fact that $C(s) = 0$ guarantees only that the fraction of 1's is at most $\alpha + \varepsilon$, and $C(s) = 1$ — that it is at least $\alpha - \varepsilon$.

3 Muchnik's Theorem

3.1 Subject Overview

An. Muchnik's theorem [9] on conditional Kolmogorov complexity states that:

Theorem 4. *Let a and b be two binary strings such that $\mathrm{C}(a) < n$ and $\mathrm{C}(a|b) < k$. Then there exists a string p such that*
- $\mathrm{C}(a|p, b) = O(\log n)$;
- $\mathrm{C}(p) \le k + O(\log n)$;
- $\mathrm{C}(p|a) = O(\log n)$.

The constants hidden in $O(\log n)$ do not depend on n, k, a, b, p.

Informally, this theorem says that there exists a program p that transforms b to a, has the minimal possible complexity $\mathrm{C}(a|b)$ (up to a logarithmic term) and, moreover, can be easily obtained from a. (The last requirement is crucial, otherwise the statement trivially reformulates the definition of conditional Kolmogorov complexity.)

Several proofs of this theorem are known. All of them rely on the existence of some combinatorial objects. The original proof in [9] leans upon the existence of bipartite graphs with specific expander-like property. Two proofs by Musatov, Romashchenko and Shen [12] use extractors and graphs allowing large on-line matchings. Explicit constructions of extractors provide variants of Muchnik's theorem for resource-bounded Kolmogorov complexity. Specifically, the following theorem is proven in [12]:

Theorem 5. *Let a and b be binary strings of length n, and k and s be integers such that $\mathrm{C}^s(a|b) < k$. Then there exists a binary string p, such that*

- $\mathrm{C}^{O(s)+\mathrm{poly}(n)}(a|p, b) = O(\log^3 n)$;
- $\mathrm{C}^s(p) \le k + O(\log n)$;
- $\mathrm{C}^{\mathrm{poly}(n)}(p|a) = O(\log^3 n)$,

where all constants in O- and poly*-notations depend only on the choice of the optimal description method.*

One cannot reduce residuals to $O(\log n)$ since all explicit extractors with such seed length output only $(1-\alpha)k$ bits, but p is taken as an output of an extractor and should have length k. However, an application of our derandomization method will decrease conditional complexities from $O(\log^3 n)$ to $O(\log n) + O(\log\log s)$ at the price of increasing the space limit in the last complexity from $\text{poly}(n)$ to $O(s) + \text{poly}(n)$. If s is polynomial in n then all space limits become also polynomial and all complexity discrepancies become logarithmic.

3.2 Proof Overview

Before we proceed with the detailed proof, let us present its high-level description. The main idea is the same in all known proofs: p is a fingerprint (or hash value) for a constructed in some specific way. This fingerprint is chosen via some underlying bipartite graph. Its left part is treated as the set of all strings of length n (i.e., all possible a's) and its right part is treated as the set of all possible fingerprints. To satisfy the last condition each left-part vertex should have small outdegree, since in that case the fingerprint is described by its number among a's neighbours. To satisfy the first condition each fingerprint should have small indegree from the strings that have low complexity conditional on b (for arbitrary b).

If resources are unbounded then the existence of a graph satisfying all conditions may be proven by the probabilistic method and the graph itself may be found by brute-force search. In the resource-bounded case we suggest to replace a random graph by a pseudo-random one and prove that it still does the job. The proof proceeds in several steps. Firstly, in Sect. 3.3 we define the essential graph property needed to our proof. We call this property low-congestion. It follows from the extractor property, but not vice versa. Secondly, in Sect. 3.4 we specify the instrumental notion of space-bounded enumerability and prove some lemmas about it in connection to low-congesting graphs. Next, in Sect. 3.5 we employ these lemmas to prove that the low-congesting property is testable by small circuits. Hence, by applying the main principle (lemma 1) this property is satisfied for pseudo-random graphs produced by the NW generator as well as for random ones. Moreover, a seed producing a graph with this property may be found in polynomial space. Finally, in Sect. 3.6 we formulate our version of Muchnik's theorem and prove it using the graph obtained on the previous step. I.e., we describe the procedure of choosing a fingerprint in this graph and then calculate all complexities and space requirements and assure that they do not exceed the respective limitations.

3.3 Low-Congesting Graphs

In fact, the proof in [12] does not use the extractor property, but employs only its corollary. In this section we accurately define this corollary in a way that allows derandomization.

Fix some bipartite graph with parameters (n, m, d) and an integer $k < n$. Let there be a system $\mathcal{S}$ of subsets of its left part with the following condition: each $S \in \mathcal{S}$ contains less than 2^k vertices, and the whole system $\mathcal{S}$ contains $2^{O(n)}$ sets (note that there are $\sum_{i=1}^{2^k} C_{2^n}^i > 2^{O(n)}$ different sets of required size, so the last limitation is not trivial). We refer to such systems as to "*relevant*" ones. Having fixed a system $\mathcal{S}$, let us call the sets in it *relevant* also.

Lemma 2. *Let $\mathcal{S}_k = \{S \subset \{0,1\}^n \mid \exists b \, \exists s \, |b| = n$ and $S = \{x \mid \mathrm{C}^s(x|b) < k\}\}$. Then the system $\mathcal{S}_k$ is relevant.*

Proof. By a standard counting argument, each set $S \in \mathcal{S}_k$ contains less than 2^k elements. If b is fixed then the described sets are expanding while s is rising. Since the largest set is also smaller than 2^k, there are less than 2^k different sets for fixed b. Since there are 2^n different b, the total size of $\mathcal{S}_k$ is bounded by $2^n 2^k = 2^{O(n)}$.

Considering a modification of the upper system $\mathcal{S}_{k,\bar{s}} = \{S \subset \{0,1\}^n \mid \exists b \, \exists s < \bar{s} \, S = \{x \mid \mathrm{C}^s(x|b) < k\}\}$, one may note that it is relevant as well.

Now fix some relevant system $\mathcal{S}$ and take an arbitrary set S in it. Call the *α-clot* for S the set of right-part vertices that have more than $\alpha DK/M$ neighbours in S (that is, at least α times more than on average). Call a vertex $x \in S$ *α-congested* (for S) if all its neighbours are in the α-clot for S. Say that G is *(α, β)-low-congesting* if there are less than βK α-congested vertices in any relevant S.

Following [2], we prove the next lemma:

Lemma 3. *Let G be an extractor graph with parameters n, m, d, k, ε. Then for any $\alpha > 1$ the graph G is (α, $\frac{\alpha}{\alpha-1}\varepsilon$)-low-congesting.*

Proof. In fact in an extractor graph there are less than $\frac{\alpha}{\alpha-1}\varepsilon K$ α-congested vertices in any set S of size K, not only in relevant ones. We may treat S as an arbitrary set of size exactly K: since a congested vertex in a subset is also a congested vertex in the set, an upper bound for the number of congested vertices in the set holds also for a subset.

Let Y be the α-clot for S, and $|Y| = \delta M$. Then the fraction $|Y|/M$ of vertices in Y equals δ and the fraction $E(S,Y)/E(S,M)$ of edges directing from S to Y is greater than $\alpha\delta$ (by the definition of clot). A standard counting argument implies only $\delta \le \frac{1}{\alpha}$, but by the extractor property $E(S,Y)/E(S,M) - |Y|/M < \varepsilon$, so $(\alpha - 1)\delta < \varepsilon$, i.e. $\delta < \frac{1}{\alpha-1}\varepsilon$. Next, let $T \subset S$ be the set of α-congested vertices in S, and $|T| = \beta K$. All edges from T direct to vertices in Y, so at least $D|T| = \beta DK$ edges direct from S to Y. In other words, the fraction of edges from S to Y is at least β. By the extractor property it must differ from the fraction of vertices in Y (that equals δ) by at most ε. So, $\beta < \delta + \varepsilon < \frac{1}{\alpha-1}\varepsilon + \varepsilon = \frac{\alpha}{\alpha-1}\varepsilon$. Putting it all together, the number of α-congested vertices in any relevant S is less than $\frac{\alpha}{\alpha-1}\varepsilon K$, so the graph is ($\alpha$, $\frac{\alpha}{\alpha-1}\varepsilon$)-low-congesting. □

3.4 Space-Bounded Enumerability

Say that a system $\mathcal{S}$ is *enumerable* in space q if there exists an algorithm with two inputs i and j working in space q that either outputs the jth element of the ith set from $\mathcal{S}$ or indicates that at least one of the inputs falls out of range. Note that for a polynomial space bound enumeration is equivalent to recognition: if one may enumerate a set then one may recognize whether a given element belongs to it by sequentially comparing it to all enumerated strings, and if one may recognize membership to a set then one may enumerate it by trying all possible strings and including only those accepted by the recognition algorithm. Only small auxiliary space is needed to perform these modifications.

Lemma 4. *The system $\mathcal{S}_{k,\bar{s}} = \{S \subset \{0,1\}^n \mid \exists b \, \exists s < \bar{s} \; |b| = n$ and $S = \{x \mid \mathrm{C}^s(x|b) < k\}\}$ is enumerable in space $O(\bar{s}) + \mathrm{poly}(n)$.*

Proof. Assume firstly that a set S is given (by specifying b and $s < \bar{s}$) and show how to enumerate it. Look through all programs shorter than k and launch them on b limiting the space to s and counting the number of steps. If this number exceeds c^s (for some constant c depending only on the computational model) then the current program has looped. In this case or if the program exceeds the space limit we terminate it and go to the next one. Otherwise, if the program produces an output, then we check whether it has not been produced by any previous program. This check is performed as follows: store the result, repeat the same procedure for all previous programs and compare their results with the stored one. If no result coincides then include the stored result in the enumeration, otherwise skip it and turn to the next program. Emulation of a program requires $O(s)$ space and no two emulations should run in parallel. All intermediate results are polynomial in n, so a total space limit $O(\bar{s}) + \mathrm{poly}(n)$ holds.

Specifying a set S by b and s is not reasonable because a lot of values of s may lead to the same set. (And if $\bar{s} = 2^{\omega(n)}$ then the number of possible indexes (b, s) exceeds the limit $2^{O(n)}$ on the size of $\mathcal{S}$.) Instead, call a limit s pivotal if $\{x \mid \mathrm{C}^s(x|b) < k\} \neq \{x \mid \mathrm{C}^{s-1}(x|b) < k\}$ and consider only pivotal limits in the definition of $\mathcal{S}_{k,\bar{s}}$. Clearly, the latter modification of definition does not affect the system itself. The advantage is that there are less than 2^k pivotal limits, and the ith pivotal limit s_i may be found algorithmically in space $O(s_i) + \mathrm{poly}(n)$. It is sufficient to construct an algorithm recognizing whether a given limit is pivotal, and the latter is done by a procedure similar to one described in the first paragraph: try all possible programs in space s, and in case they produce an output, check whether it is produced by any program in space $s - 1$. If at least one result is new then s is pivotal, otherwise it is not. Putting all together, a set S in $\mathcal{S}_{k,\bar{s}}$ is defined by a word b and the ordinal number i of a pivotal space limit s_i. Knowing these parameters, one may enumerate it in space $O(\bar{s}) + \mathrm{poly}(n)$, as claimed.

The next lemma indicates that if a system $\mathcal{S}$ is enumerable in small space, then the same holds for the system of congested subsets of its members. We call a bipartite graph *computable* in space q if there exists an algorithm working in space q that receives an index of a left-part vertex and an index of an incident edge and outputs the right-part vertex this edge directs to.

Lemma 5. *Let $\mathcal{S}$ be a system of relevant sets enumerable in space s, G be a bipartite graph computable in space q and $\alpha > 1$ be a (rational) number. Then the system $\mathrm{Con}_\alpha \mathcal{S} = \{T \mid \exists S \in \mathcal{S}$ for which T is the set of α-congested vertices$\}$ is computable in space $O(\max\{s, q\}) + \mathrm{poly}(n)$. Moreover, the iteration $(\mathrm{Con}_\alpha)^r \mathcal{S}$ is also computable in space $O(\max\{s, q\}) + \mathrm{poly}(n)$ with constants in O- and* poly- *notations not depending on r, but possibly depending on α.*

Proof. Since for space complexity enumeration and recognition are equivalent, it is sufficient to recognize that a vertex x is α-congested. The recognizing algorithm works as follows: having a set S and a vertex $x \in S$ fixed, search through all neighbours of x (this requires space $O(q) + \text{poly}(n)$) and for each neighbour check whether it lies in the α-clot for S. If all neighbours do then x is congested, otherwise it is not. Having a neighbour y fixed, the check is performed in the following way: enumerate all members of S (using $O(s) + \text{poly}(n)$ space), for each member search through all its neighbours (using $O(q) + \text{poly}(n)$ space) and count the number of these neighbours coinciding with y. Finally, compare this number with the threshold $\alpha DK/M$. Note that no two computations requiring space s or q run in parallel and all intermediate results need only polynomial space, so the total space requirement is $O(\max\{s, q\}) + \text{poly}(n)$, as claimed. Note also that O- notation is used only due to the possibility of computational model change, not because of the necessity of looping control. If a computational model is fixed then the required space is just $\max\{s, q\} + \text{poly}(n)$.

The last observation is crucial for the "moreover" part of lemma. Indeed, by a simple counting argument the fraction of α-congested vertices in S is at most $1/\alpha$. That is why after at most $\log_\alpha 2^n = O(n)$ iterations the set of congested variables becomes empty. Each iteration adds $\text{poly}(n)$ to the space requirement, so the overall demand is still $\max\{s, q\} + \text{poly}(n)$ (with greater polynomial), as claimed.

3.5 Derandomization

In this section we show that, firstly, the low-congesting property may be (approximately) recognized by $2^{O(n)}$-sized constant-depth circuits, secondly, that there are low-congesting graphs in the output of Nisan-Wigderson pseudo-random generator and, thirdly, that one can recognize in polynomial space whether the NW-generator produces a low-congesting graph on a given seed. Put together, the last two lemmas provide a seed on which the NW-generator produces a low-congested graph.

Let us encode a bipartite graph with parameters (n, m, d) by a list of edges. The length of this list is $2^n 2^d m$: for each of 2^n left-part vertices we specify 2^d neighbours, each being m-bit long.

Lemma 6. *Let $\mathcal{G}$ be the set of all bipartite graphs with parameters (n, m, d) encoded as described above. Let k be an integer such that $1 < k < n$, and $\varepsilon > 0$. Then there is a circuit C of size $2^{O(n)}$ and constant depth, defined on $\mathcal{G}$, such that:*

- *If G is a (k, ε)-extractor then $C(G)=1$;*
- *If $C(G) = 1$ then G is a $(2.01, 2.01\varepsilon)$-low-congesting graph for $\mathcal{S}_k$ from lemma 2.*

Proof. We build a non-uniform circuit, so we may assume that $\mathcal{S}_k$ is given. We construct a single circuit approximately counting the number of congested vertices in a given set, then replicate it for each relevant set and take conjunction.

Since there are less than 2^{n+k} relevant sets, this operation keeps the size of circuit being $2^{O(n)}$. We proceed by constructing a circuit for a given set S.

The size of the input is $|S| \cdot 2^d m$. We think of it as of being divided into $|S|$ blocks of 2^d segments of length m. We index all blocks with elements $x \in S$ and index all segments of the block x by vertices y incident to x. It is easy to see that there is a constant-depth circuit that compares two segments (that is, has $2m$ inputs and outputs 1 if and only if the first half of inputs coincides with the second half). On the first stage we apply this circuit to every pair of segments, obtaining a long 0-1-sequence. On the second stage we employ a counting circuit with $|S|D-1$ arguments that is guaranteed to output 1 if more than $2.01DK/M$ of its arguments are 1's and to output 0 if the number of 1's is less than $2DK/M$. By theorem 3 there exists such a circuit of polynomial (in the number of arguments) size and constant depth. For all segments y a copy of this circuit is applied to the results of the comparison of y to all other segments. If y lies in the 2.01-clot then the respective copy outputs 1, and if it outputs 1, then y lies in 2-clot. On the third stage we take a conjuction of all second-stage results for the segments lying in the same block x. If this conjunction equals 1 then all images of x lie in 2-clot, that is, x is 2-congested. Conversely, if x is 2.01-congested then the conjunction equals 1. Finally, we utilize another counting circuit with $|S|$ inputs that outputs 0 if *more* than $2.01\varepsilon K$ of its inputs are 1's and outputs 1 if *less* than $2\varepsilon K$ of its inputs are 1's. This circuit is applied to all outputs of the third stage. If the final result is 1 then less than $2.01\varepsilon K$ elements of S are 2.01-congested; and if less than $2\varepsilon K$ elements of S are 2-congested then the final result is 1.

The last claim holds for any relevant S. On the very last stage we take a conjunction of results for all S. If this conjunction is 1 then less than $2.01\varepsilon K$ elements in each S are 2.01-congested, and if G is a (k, ε)-extractor then by lemma 3 less than $2\varepsilon K$ elements in each S are 2-congested and hence this conjunction equals 1.

Since there are at most $2^{O(n)}$ gates on every stage and each stage has constant depth, the overall circuit has also $2^{O(n)}$ gates and constant depth, as claimed.

Lemma 7. *Let n, $m = k$, $d = O(\log n)$ and ε be such parameters that a random bipartite graph with parameters $(n,\ d,\ m)$ is a $(k,\ \varepsilon)$-extractor with constant probability $p > 0$. Let q be the depth of the circuit from the previous lemma. Let $NW\colon \{0.1\}^l \to \{0,1\}^N$, where $l = O(\log^{2q+6})$ and $N = 2^n 2^d m$, be the Nisan-Wigderson generator from corollary 1. Then $\mathrm{Prob}_u\{C(NW(u)) = 1\} > \frac{p}{2}$ for sufficiently large n, where C is the circuit from the previous lemma.*

Proof. This is a straightforward application of lemma 1. By the previous lemma if a graph G is an extractor then $C(G) = 1$, so $\mathrm{Prob}_G\{C(G) = 1\} \geq p$. Since C is a constant-depth circuit, the property $C(G) = 1$ is tautologically testable by a constant-depth circuit. By lemma 1 $\mathrm{Prob}_u\{C(NW(u)) = 1\} > \frac{p}{2}$. □

Consider the following problem $\mathcal{R}$: find a string $u \in \{0,1\}^l$ such that the graph $NW(u)$ is $(2.01,\ 2.01\varepsilon K)$-low-congesting with respect to $\mathcal{S}_{k,\,s}$.

Lemma 8. *The problem $\mathcal{R}$ is solvable in space $O(s) + \mathrm{poly}(n)$.*

Proof. The existence of a solution follows from the previous lemma. Since we care only about the space limit, the search problem may be replaced by the corresponging recognition problem. The space bound for the latter one arises from corollary 1, lemma 4 and lemma 5. Indeed, by corollary 1 the graph $NW(u)$ is computable in polynomial space, and by lemma 4 the system $\mathcal{S}_{k,\,s}$ is enumerable in space $O(s) + \mathrm{poly}(n)$. Hence by lemma 5 the system $\mathrm{Con}_{2.01}\,\mathcal{S}_{k,\,s}$ is also enumerable in space $O(s)+\mathrm{poly}(n)$, therefore one may easily check whether each set in $\mathrm{Con}_{2.01}\,\mathcal{S}_{k,\,s}$ contains less than $2.01\varepsilon K$ elements, thus solving the recognition analogue of $\mathcal{R}$. Only polynomial extra space is added on the last step.

3.6 Proof of the Theorem

Now we proceed by formulating and proving our version of Muchnik's theorem.

Theorem 6. *Let a and b be binary strings of length less than n, and s and k be numbers such that $\mathrm{C}^s(a|b) < k$. Then there exists a binary string p, such that*

- $\mathrm{C}^{O(s)+\mathrm{poly}(n)}(a|p, b) = O(\log\log s + \log n)$*;*
- $\mathrm{C}^s(p) \le k + O(\log n)$*;*
- $\mathrm{C}^{O(s)+\mathrm{poly}(n)}(p|a) = O(\log\log s + \log n)$*,*

where all constants in O- and poly*-notations depend only on the choice of the optimal description method.*

Proof. Basically the proof proceeds as the respective proof of theorem 5 in [12]. Here we replace an explicitly constructed extractor by a pseudorandom graph described in Sect. 3.5.

Let ε be a small constant and let $d = O(\log n)$ be such that a random bipartite graph with parameters $(n, m = k, d)$ is a (k, ε)-extractor with probability greater than some positive constant μ. Let l be a parameter and NW be a function from lemma 7. By lemmas 7 and 6 the output of NW is a $(2.01, 2.01\varepsilon K)$-low-congesting graph for $\mathcal{S}_k$ with probability at least $\mu/2$. Applying the program from lemma 8, one may find in $O(s) + \mathrm{poly}(n)$ space a seed u for which $NW(u)$ is a $(2.01, 2.01\varepsilon K)$-low-congesting graph for $\mathcal{S}_{k,\,s}$. This u has low complexity: to perform this search one needs to know parameters n, k and $l = \mathrm{poly}(n)$, that is, $O(\log n)$ bits, and the space bound s, that requires $\log s$ bits. The last number may be reduced to $\log\log s$, because the space bound s may be replaced by the least power of 2 exceeding s keeping the needed space to be $O(s) + \mathrm{poly}(n)$. For what follows, fix this seed u and the graph $G = NW(u)$.

By definition of the low-congesting property the number of 2.01-congested vertices in the set $S = \{x \mid \mathrm{C}^s(x|b) < k\}$ does not exceed $2.01\varepsilon K$. Clearly, a belongs to S. Firstly suppose that it is not 2.01-congested. Then it has a neighbour outside of the 2.01-clot for S. This neighbour may be taken as p. Indeed, the length of p equals $m = k$, hence its complexity is also less than $k + O(1)$. To specify p knowing a, one needs to construct the graph $NW(u)$, for which only $O(\log n) + O(\log\log s)$ bits are necessary, and to know the number of p among a's neighbours, which is at most $d = O(\log n)$. Finding a given b

and p proceeds as follows: construct G, enumerate S and choose the specified preimage of p in S. Then a is determined by the information needed to construct G ($O(\log n + \log\log s)$ bits), information needed to enumerate S (that is, k, $\log s$ and b) and the number of a among preimages of p (since p is not in the 2.01-clot, there are not more than $2.01DK/M = 2.01D$ preimages, so $O(d) = O(\log n)$ bits are required). Summarizing, we get $O(\log n) + O(\log\log s)$ bits and $O(s) + \text{poly}(n)$ space needed. Note that the fact that $m = k$ is crucial here: in the case $m = (1-\alpha)k$ the number of required bits would increase by αk and exceed the bound.

Now we turn to the case where a *is* 2.01-congested. By lemma 5 the set $\text{Con}_{2.01}\, S$ of 2.01-congested vertices is enumerable in $O(s) + \text{poly}(n)$ space. Take parameters $n_1 = n$, $m_1 = k_1 = \log K_1 = \log(2.01\varepsilon K)$, $d_1 = O(\log n)$ and $\varepsilon_1 = \varepsilon$ such that an extractor with these parameters exists with probability greater than μ. Lemmas 6, 7 and 8 are applicable to the new situation, so we may find a new u_1 such that the graph $G_1 = NW(u_1)$ is $(2.01,\, 2.01\varepsilon K_1)$-low-congesting for $\text{Con}_{2.01}\, \mathcal{S}_{k,\,s}$ with probability at least $\mu/2$. By assumption, a belongs to the set $\text{Con}_{2.01}\, S$. If it is not 2.01-congested in it then we choose p analogously to the initial situation. Otherwise we reduce the parameters again and take a low-congesting graph for $\text{Con}^2_{2.01}\, \mathcal{S}_{k,\,s}$, and so on. By the "moreover" part of lemma 5, this reduction may be performed iteratively for arbitrary number of times keeping the space limit to be $O(s) + \text{poly}(n)$.

The total number of iterations is less than $\log_{1/2.01\varepsilon} k = O(\log n)$. Finally p is defined as a neighbour of a not lying in the 2.01-clot in graph G_i. To find p knowing a one needs to know i and the same information as on the upper level. To find a knowing p and b also only specifying i is needed besides what has been specified on the upper level. So, all complexities and space limits are as claimed and the theorem is proven.

Acknowledgments

I want to thank my colleagues and advisors Andrei Romashchenko, Alexander Shen and Nikolay Vereshchagin for stating the problem and many useful comments. I also want to thank three anonymous referees for careful reading and precise comments. I am grateful to participants of seminars in Moscow State University and Moscow Institute for Physics and Technology for their attention and thoughtfulness.

References

1. Ajtai, M.: Approximate counting with uniform constant-depth circuits. In: Advances in computational complexity theory, pp. 1–20. American Mathematical Society, Providence (1993)
2. Buhrman, H., Fortnow, L., Laplante, S.: Resource bounded Kolmogorov complexity revisited. SIAM Journal on Computing 31(3), 887–905 (2002)
3. Buhrman, H., Lee, T., van Melkebeek, D.: Language compression and pseudorandom generators. In: Proc. of the 15th IEEE Conference on Computational Complexity. IEEE, Los Alamitos (2004)

4. Dvir, Z., Wigderson, A.: Kakeya sets, new mergers and old extractors. In: Proceedings of the 49th Annual FOCS 2008, pp. 625–633 (2008)
5. Guruswami, V., Umans, C., Vadhan, S.: Unbalanced expanders and randomness extractors from Parvaresh-Vardy codes. Journal of the ACM 56(4) (2009)
6. Li, M., Vitanyi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 2nd edn. Springer, Heidelberg (1997)
7. Lu, C.-J., Reingold, O., Vadhan, S., Wigderson, A.: Extractors: Optimal up to Constant Factors. In: 35th Annual ACM Symposium, STOC 2003, pp. 602–611 (2003)
8. Muchnik, A.A.: On basic structures of the descriptive theory of algorithms. Soviet Math. Dokl. 32, 671–674 (1985)
9. Muchnik, A.: Conditional complexity and codes. Theoretical Computer Science 271(1-2), 97–109 (2002)
10. Musatov, D.: Extractors and an effective variant of Muchnik's theorem. Diplom (Master thesis).Faculty of Mechanics and Mathematics, MSU (2006) (in Russian), http://arxiv.org/abs/0811.3958
11. Musatov, D.: Improving the space-bounded version of Muchnik's conditional complexity theorem via "naive" derandomization (2010), http://arxiv.org/abs/1009.5108 (Online version of this paper)
12. Musatov, D., Romashchenko, A., Shen, A.: Variations on muchnik's conditional complexity theorem. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) CSR 2009. LNCS, vol. 5675, pp. 250–262. Springer, Heidelberg (2009)
13. Musatov, D., Romashchenko, A., Shen, A.: Variations on Muchnik's conditional complexity theorem, to be published in TOCS
14. Nisan, N.: Pseudorandom bits for constant depth circuits. Combinatorica 11(1), 63–70 (1991)
15. Nisan, N., Wigderson, A.: Hardness vs. Randomness. Journal of Computer and System Sciences 49, 149–167 (1994)
16. Radhakrishnan, J., Ta-Shma, A.: Bounds for dispersers, extractors, and depth-two superconcentrators. SIAM Journal on Discrete Mathematics 13(1), 2–24 (2000)
17. Reingold, O., Shaltiel, R., Wigderson, A.: Extracting randomness via repeated condensing. SIAM Journal on Computing 35(5), 1185–1209 (2006)
18. Romashchenko, A.: Pseudo-random graphs and bit probe schemes with one-sided error. In: CSR 2011
19. Slepian, D., Wolf, J.K.: Noiseless coding of correlated information sources. IEEE Transactions on information Theory 19, 471–480 (1973)
20. Trevisan, L.: Construction of extractors using pseudo-random generators. In: Proc. 45th Annual Symposium on Foundations of Computer Science, pp. 264–275
21. Zvonkin, A., Levin, L.: 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)

The Complexity of Solving Reachability Games Using Value and Strategy Iteration*

Kristoffer Arnsfelt Hansen, Rasmus Ibsen-Jensen, and Peter Bro Miltersen

Aarhus University
{arnsfelt,rij,pbmiltersen}@cs.au.dk

Abstract. Two standard algorithms for approximately solving two-player zero-sum concurrent reachability games are *value iteration* and *strategy iteration*. We prove upper and lower bounds of $2^{m^{\Theta(N)}}$ on the worst case number of iterations needed for both of these algorithms to provide non-trivial approximations to the value of a game with N non-terminal positions and m actions for each player in each position.

1 Introduction

1.1 Statement of Problem and Overview of Results

We consider *finite state, two-player, zero-sum, deterministic, concurrent reachability games*. For brevity, we shall henceforth refer to these as just reachability games. The class of reachability games is a subclass of the class of games dubbed *recursive games* by Everett [8] and was introduced to the computer science community in a seminal paper by de Alfaro, Henzinger and Kupferman [6]. A reachability game G is played between two players, Player I and Player II. The game has a finite set of *non-terminal* positions and special *terminal* positions GOAL and TRAP. In this paper, we let N denote the number of non-terminal positions and assume positions are indexed $1, \ldots, N$ while GOAL is indexed $N + 1$ and TRAP 0. At any point in time during play, a *pebble* rests at some position. The position holding the pebble is called the *current* position. The objective for Player I is to eventually make the current position GOAL. If this happens, play ends and Player I wins. The objective for Player II is to *forever* prevent this from happening. This may be accomplished either by the pebble reaching TRAP from where it cannot escape or by it moving between non-terminal positions indefinitely. To each non-terminal position i is associated a finite set of actions A_i^1, A_i^2 for each of the two players. In this paper, we assume that all these sets have the same size m (if not, we may "copy" actions to make this so) and that $A_i^1 = A_i^2 = \{1, \ldots, m\}$. At each point in time, if the current position is i, Player I and Player II simultaneously choose actions in $\{1, \ldots, m\}$. For each position i and each action pair $(a, a') \in \{1, \ldots, m\}^2$ is

* Work supported by Center for Algorithmic Game Theory, funded by the Carlsberg Foundation. The authors acknowledge support from The Danish National Research Foundation and The National Science Foundation of China (under the grant 61061130540) for the Sino-Danish Center for the Theory of Interactive Computation, under which part of this work was performed.

associated a position $\pi(i, a, a')$. In other words, each position holds an $m \times m$ matrix of *pointers* to positions. When the current position at time t is i and the players play the action pair (a, a'), the new position of the pebble at time $t+1$ is $\pi(i, a, a')$.

A *strategy* for a reachability game is a (possibly randomized) procedure for selecting which action to take, given the history of the play so far. A strategy *profile* is a pair of strategies, one for each player. A *stationary strategy* is the special case of a strategy where the choice only depends on the current position. Such a strategy is given by a family of probability distributions on actions, one distribution for each position, with the probability of an action according to such a distribution being called a *behavior probability*. We let $\mu_i(x, y)$ denote the probability that Player I eventually reaches GOAL if the players play using the strategy profile (x, y) and the pebble starts in position i. The *lower value* of position i is defined as: $\underline{v_i} = \sup_{x \in S^1} \inf_{y \in S^2} \mu_i(x, y)$ where S^1 (S^2) is the set of strategies for Player I (Player II). Similarly, the *upper value* of a position i is $\overline{v_i} = \inf_{y \in S^2} \sup_{x \in S^1} \mu_i(x, y)$. Everett [8] showed that for all positions i in a reachability game, the lower value $\underline{v_i}$ in fact equals the upper value $\overline{v}_i$, and this number is therefore simply called the *value* v_i of that position. The vector v is called the *value vector* of the game. Furthermore, Everett showed that for any $\epsilon > 0$, there is a stationary strategy x^* of Player I so that for all positions i, we have $\inf_{y \in S^2} \mu_i(x^*, y) \geq v_i - \epsilon$, i.e. the strategy x^* guarantees the value of any position within ϵ when play starts in that position. Such a strategy is called *ϵ-optimal*. Note that x^* does not depend on i. It may however depend on $\epsilon > 0$ and this dependence may be necessary, as shown by examples of Everett. In contrast, it is known that Player II has an exact optimal strategy that is guaranteed to achieve the value of the game, without any additive error [17,13].

In this paper, we consider algorithms for *solving* reachability games. There are two notions of solving a reachability game relevant for this paper:

1. *Quantitatively*: Given a game, compute ϵ-approximations of the entries of its value vector (we consider approximations, rather than exact computations, as the value of a reachability game may be an irrational number).
2. *Strategically*: Given a game, compute an ϵ-optimal strategy for Player I.

Once a game has been solved strategically, it is straightforward to also solve it quantitatively (for the same ϵ) by analyzing, using linear programming, the finite state Markov decision process for Player II resulting when freezing the computed strategy for Player I. The converse direction is far from obvious, and it was in fact shown by Hansen, Koucký and Miltersen [12] that if standard binary representation of behavior probabilities is used, merely *exhibiting* an $(1/4)$-optimal strategy requires worst case exponential space in the size of the game. In contrast, a $(1/4)$-approximation to the value vector obviously only requires polynomial space to describe and it may be possible to compute it in polynomial time, though it is currently not known how to do so [5].

There is a large and growing literature on solving reachability games [6,7,3,1,2,12]. In this paper, we focus on the two perhaps best-known and best-studied algorithms, *value iteration* and *strategy iteration*. Both were originally derived from similar algorithms for solving Markov decision processes [15] and *discounted* stochastic games [19]. We describe these algorithms next. Value iteration is Algorithm 1. Value iteration approximately solves reachability games quantitatively.

Algorithm 1. Value Iteration

```
1: $t := 0$
2: $\tilde{v}^0 := (0, \ldots, 0, 1)$ // the vector $\tilde{v}^0$ is indexed $0, 1, \ldots, N, N+1$
3: while true do
4:     $t := t + 1$
5:     $\tilde{v}^t_0 := 0$
6:     $\tilde{v}^t_{N+1} := 1$
7:     for $i \in \{1, 2, \ldots, N\}$ do
8:         $\tilde{v}^t_i := \text{val}(A_i(\tilde{v}^{t-1}))$
```

Algorithm 2. Strategy Iteration

```
1: $t := 1$
2: $x^1$ := the strategy for Player I playing uniformly at each position
3: while true do
4:     $y^t$ := an optimal best reply by Player II to $x^t$
5:     for $i \in \{0, 1, 2, \ldots, N, N+1\}$ do
6:         $v^t_i := \mu_i(x^t, y^t)$
7:     $t := t + 1$
8:     for $i \in \{1, 2, \ldots, N\}$ do
9:         if $\text{val}(A_i(v^{t-1})) > v^{t-1}_i$ then
10:            $x^t_i := \text{maximin}(A_i(v^{t-1}))$
11:        else
12:            $x^t_i := x^{t-1}_i$
```

In the pseudocode of Algorithm 1, the matrix $A_i(\tilde{v}^{t-1})$ denotes the result of replacing each pointer to a position j in the $m \times m$ matrix of pointers at position i with the real number $\tilde{v}^{t-1}_j$. That is, $A_i(\tilde{v}^{t-1})$ is a matrix of $m \times m$ real numbers. Also, $\text{val}(A_i(\tilde{v}^{t-1}))$ denotes the value of the *matrix game* with matrix $A_i(\tilde{v}^{t-1})$ and the row player being the maximizer. This value may be found using linear programming. Value iteration works by iteratively updating a *valuation* of the positions, i.e., the numbers $\tilde{v}^t_i$. Clearly, when implementing the algorithm, valuations $\tilde{v}^t_i$ only have to be kept for one iteration of the while loop after the iteration in which they are computed and the algorithm thus only needs to store $O(N)$ real numbers.[1] As stated, the algorithm is non-terminating, but has the property that as t approaches infinity, the valuations $\tilde{v}^t_i$ approach the correct values v_i from below. We present an easy (though not self-contained) proof of this well-known fact in section 2.1 below, and also explain the intuition behind the truth of this statement. However, until the present paper, there has been no published information on the number of iterations needed for the approximation to be an ϵ-approximation to the correct value for the general case of concurrent reachability

[1] In this paper, we assume the real number model of computation and ignore the (severe) technical issues arising when implementing the algorithm using finite-precision arithmetic.

games, though Condon [4] observed that for the case of *turn-based* games (or "simple stochastic games"), the number of iterations has to be at least exponential in N in order to achieve an ϵ-approximation. Clearly, the concurrent case is at least as bad. In fact, this paper will show that the concurrent case is much worse!

Strategy iteration is Algorithm 2. It approximately solves reachability games quantitatively as well as strategically. In the pseudocode of Algorithm 2, the line "$y^t :=$ an optimal *best reply* to x^t" should be interpreted as follows: When Player I's strategy has been "frozen" to x^t, the resulting game is a one-player game for Player II, also known as an *absorbing Markov decision process*. For such a process, an optimal stationary strategy y^t that is pure is known to exist, and can be found in polynomial time using linear programming [15]. The expression $\text{maximin}(A_i(v^{t-1}))$ denotes a maximin mixed strategy (an "optimal strategy") for the maximizing row player in the matrix game $A_i(v^{t-1})$. This optimal strategy may again be found using linear programming. The strategy iteration algorithm was originally described for *one-player games* by Howard [15], with Player I being the single player – in that case, in the pseudocode, the line "$y^t :=$ an optimal *best reply* to x^t" is simply omitted. Subsequently, a variant of the pseudocode of Algorithm 2 was shown by Hoffman and Karp [14] to be a correct approximation algorithm for the class of *recurrent undiscounted stochastic games* and by Rao, Chandrasekaran and Nair [18] to be a correct algorithm for the class of *discounted stochastic games*. Finally, Chatterjee, de Alfaro and Henzinger [1] showed the pseudocode of Algorithm 2 to be a correct approximation algorithm for the class of reachability games. As is the case for value iteration, the strategy iteration algorithm is non-terminating, but has the property that as t approaches infinity, the valuations v_i^t approach the correct values v_i from below. Chatterjee *et al.* [1, Lemma 8] prove this by relating the algorithm to the value iteration algorithm. In particular, they prove:

$$\tilde{v}_i^t \leq v_i^t \leq v_i. \tag{1}$$

That is, strategy iteration needs at most as many iterations of the while loop as value iteration to achieve a particular degree of approximation to the correct values v_i. Also, the strategies x^t *guarantee* the valuations v_i^t for Player I, so whenever these valuations are ϵ-close to the values, the corresponding x^t is an ϵ-optimal strategy. However, until the present paper, there has been no published information on the number of iterations needed for the approximation to be an ϵ-optimal solution, though a recent breakthrough result of Friedman [9] proved that for the case of *turn-based* games, the number of iterations is at least exponential in N in the worst case. Clearly, the concurrent case is at least as bad. In fact, this paper will show that the concurrent case is much worse!

As our main result, we exhibit a family of reachability games with N positions and m actions for each player in each position, such that all non-terminal positions have value one and such that value iteration as well as strategy iteration need at least a *doubly exponential* $2^{m^{\Omega(N)}}$ number of iterations to obtain valuations larger than any fixed constant (say 0.01). By inequality (1), it is enough to consider the strategy iteration algorithm to establish this. However, our proof is much easier and cleaner for the value iteration algorithm, the exact bounds are somewhat better, and our much more technical proof for the strategy iteration case is in fact based upon it. So, we shall present separate proofs for the two cases, and in these proceedings, due to lack of space, only with details for the first case.

Our hard instances $P(N, m)$ for both algorithms are generalizations of the "Purgatory" games defined by Hansen, Miltersen and Koucký [12] (these occur as special cases by setting $m = 2$). Following the conventions of that paper, we describe these games as being games between *Dante* (Player I) and *Lucifer* (Player II). The game $P(N, m)$ can be described succinctly as follows: *Lucifer repeatedly selects and hides a number between 1 and m. Each time Lucifer hides such a number, Dante must try to guess which number it is. After the guess, the hidden number is revealed. If Dante ever guesses a number which is strictly higher than the one Lucifer is hiding, Dante loses the game. If Dante ever guesses correctly N times in a row, the game ends with Dante being the winner. If neither of these two events ever happen and the play thus continues forever, Dante loses.* It is easy to see that $P(N, m)$ can be described as a deterministic concurrent reachability game with N non-terminal positions and m actions for each player in each position. Also, by applying a polynomial-time algorithm by de Alfaro *et al.* [6] for determining which positions in a reachability game have value 1, we find that all positions except TRAP have value 1 in $P(N, m)$. That is, Dante can win this game with arbitrarily high probability.

We note that these hard instances are very natural and easy to describe "as games" that one might even conceivably have a bit of fun playing (the reader is invited to try playing $P(2, 2)$ with an uninitiated party)! In this respect, they are quite different from the recent extremely ingenious turn-based games due to Friedman [9] where strategy iteration exhibits exponential behavior.

Using recent improved upper bounds on the *patience* of ϵ-optimal strategies for Everett's recursive games, we provide matching $2^{m^{O(N)}}$ upper bounds on the number of iterations sufficient for getting adequate approximate values, by each of the algorithms. In particular, both algorithms are also of *at most* doubly-exponential complexity.

Table 1. Running Strategy Iteration on $P(7, 2)$

# Iterations	10^0	10^1	10^2	10^3	10^4	10^5	10^6	10^7	10^8
Valuation	0.01347	0.03542	0.06879	0.10207	0.13396	0.16461	0.19415	0.22263	0.24828

That the doubly-exponential complexity is a real phenomenon is illustrated in Table 1 which tabulates the valuations computed by strategy iteration for the initial position of $P(7, 2)$, i.e., "Dante's Purgatory" [12], a 7-position game of value 1. The algorithm was implemented using floating point arithmetic and was allowed to run for one hundred million iterations at which point the precision was inadequate for representing the computed strategies (note that the main result of Hansen, Miltersen and Koucký [12] implies that roughly 64 *decimal* digits of precision is needed to describe a strategy achieving a valuation above 0.9).

Interestingly, when introduced as an algorithm for solving concurrent reachability games [1], strategy iteration was proposed as a practical alternative to generic algorithms having an exponential worst case complexity. More precisely, one obtains a generic algorithm for solving reachability games quantitatively by reducing the problem to the decision problem for the existential fragment of the first order theory of the real numbers [7]. This yields an exponential time (in fact a **PSPACE**) algorithm. Our results show that this generic algorithm is in fact astronomically more practical than strategy iteration on very simple and natural instances. Still, it is not practical in any real sense of this term, even given state-of-the-art implementations of the best known decision procedures for the theory of the reals. Finding a practical algorithm remains a very interesting open problem.

1.2 Overview of Proof Techniques

Our proof of the lower bound for the case of value iteration is very intuitive. It is based on combining the following facts:

1. The valuations $\tilde{v}_i^t$ obtained in iteration t of value iteration is in fact the values of a *time bounded* version of the reachability game, where Player I loses if he has not reached GOAL at time t.
2. While the value of the game $P(N, m)$ is 1, the value of its time bounded version is very close to 0 for all small values of t.

The second fact was established by Hansen *et al.* [12] for the case $m = 2$ by relating the so-called *patience* of reachability games to the values of their time bounded version, without the connection to the value iteration algorithm being made explicit, by giving bounds on the patience of the games $P(N, 2)$. The present paper provides a different and arguably simpler proof of the lower bound on the value of the time bounded game that gives bounds also for other values of m than 2. It is based on exhibiting a fixed strategy for Lucifer that prevents Dante from winning fast.

The lower bound for strategy iteration is much more technical. We remark that the analysis of value iteration is used twice and in two different ways in the proof. It proceeds roughly as follows: The analysis of value iteration yields that when value iteration is applied to $P(1, m)$, exponentially many iterations (in m) are needed to yield a close approximation of the value. We can also show that when strategy iteration is applied to $P(1, m)$, exactly the same sequence of valuations is computed as when value iteration is applied to the same game. From these two facts, we can derive an upper bound on the patience of the strategies computed by strategy iteration on $P(1, m)$. Next, a quite involved argument shows that when applying strategy iteration to $P(N, m)$, the sequence of strategies computed for one of the positions (the initial one) is exactly the same as the one computed when strategy iteration is applied to $P(1, m)$. We also show that the smallest behavior probability in the computed strategy for $P(N, m)$ occurs in the initial position. In particular, the patiences of the sequence of strategies computed for $P(N, m)$ is the same as the patiences of the sequence of strategies computed for $P(1, m)$. Finally, our analysis of value iteration for $P(N, m)$ and the relationship between patience and value iteration allow us to conclude that a strategy with low patience for $P(N, m)$ cannot be near-optimal, yielding the desired doubly-exponential lower bound.

2 Theorems and Proofs

2.1 The Connection between Patience, the Value of Time Bounded Games, and the Complexity of Value Iteration

The key to understanding value iteration is the following folklore lemma. Given a concurrent reachability game G, we define G_T to be the *finite* extensive form game with the same rules as G, except that Player 1 loses if he has not reached GOAL after T moves of the pebble. The positions of G_T are denoted by (i, t), where i is a position of G and t is an integer denoting the number of time steps left until Dante's time is out.

Lemma 1. *The valuation $\tilde{v}_i^t$ computed by the value iteration algorithm when applied to a game G is the exact value of position (i, t) in the game G_t.*

The proof is an easy induction in t ("Backward induction"). A very general result by Mertens and Neyman [16] establishes that for a much more general class of games (undiscounted stochastic games), the value of the time bounded version converges to the value of the infinite version as the time bound approaches infinity. Combining this with Lemma 1 immediately yields the correctness of the value iteration algorithm.

The *patience* [8] of a stationary strategy for a concurrent reachability game is $1/p$, where p is the smallest non-zero behavior probability employed by the strategy in any position. The following lemma relates the patience of near-optimal strategies of a reachability game to the difference between the values of the time bounded and the infinite game and hence to the convergence rate of value iteration.

Lemma 2. *Let G be a reachability game with N non-terminal positions and with an ϵ-optimal strategy of patience at most l, for some $l \geq 1, \epsilon > 0$. Let $T = kNl^N$ for some $k \geq 1$, and u be any position of G. Then, the value of position (u, T) of G_T differs from the value of the position u of G by at most $\epsilon + e^{-k}$.*

Proof. We want to show that the value of (u, T) in G_T is at least $v_u - \epsilon - e^{-k}$, where v_u is the value of position u in G. We can assume that $v_u > \epsilon$, because otherwise we are done. Fix an ϵ-optimal stationary strategy x for Dante in G of patience at most l. Consider this as a strategy of G_T and consider play starting in u. We shall show that x guarantees Dante to win G_T with probability at least $v_u - \epsilon - e^{-k}$, thus proving the statement. Consider a best reply y by Lucifer to x in G_T. Note that y does not necessarily correspond to a stationary strategy in G. The strategy can still be played by Lucifer in G, by playing by it for the first T time steps and playing arbitrarily afterwards.

Call a position v of G *alive* if there are paths from v to GOAL in *all* directed graphs obtained from G in the following way: The nodes of the graphs are the positions of G. We then select for each position an arbitrary column for the corresponding matrix, and let the edges going out from this node correspond to the pointers of the chosen column and rows where Dante assigns positive probability. That is, intuitively, a position v is alive, if and only if there is no *absolutely sure* way for Lucifer for preventing Dante from reaching GOAL when play starts in v. Positions that are not alive are called *dead*. Note that if a position v is dead, the strategy y, being a best reply of Lucifer, will pick actions so that the probability of play reaching GOAL, conditioned on play having reached v, is 0. On the other hand, if the current position v is alive, the conditional probability that play reaches GOAL within the next N steps is at least $(1/l)^N$. That is, looking at the entire play, the probability that play has *not* reached either GOAL or a dead state after T steps is at most $(1-l^{-N})^{T/N} = (1-l^{-N})^{kl^N} \leq e^{-k}$. Suppose now that GOAL is reached in T steps with probability strictly less than $v_u - \epsilon - e^{-k}$ when play starts in u. This means that a dead position is reached with probability strictly greater than $1-(v_u-\epsilon-e^{-k})-e^{-k}$, i.e., strictly greater than $1-(v_u-\epsilon)$. But this means that if Lucifer plays y as a reply to x in the *infinite* game G he will in fact succeed in getting the pebble to reach a dead position and hence prevent Dante from *ever* reaching GOAL, with probability strictly greater than $1-(v_u-\epsilon)$. This contradicts x being ϵ-optimal for Dante in G. Thus, we conclude that GOAL is in fact reached in T steps with probability at least $v_u - \epsilon - e^{-k}$ when play starts in u with x and y being played against each other in G_T, as desired.

The connection between the convergence of value iteration and the time bounded version of the game allows us to reformulate the lemma in the following very useful way.

Lemma 3. *Let G be a reachability game with an ϵ-optimal strategy of patience at most l, for some $\epsilon > 0$. Then, $T = kNl^N$ rounds of value iteration is sufficient to approximate the values of all positions of the game with additive error at most $\epsilon + e^{-k}$.*

We can use this lemma to prove our upper bound on the number of iterations of value iteration (and hence also strategy iteration). The following lemma is from Hansen *et al.* [11].

Lemma 4 (Hansen, Koucký, Lauritzen, Miltersen and Tsigaridas). *Let $\epsilon > 0$ be arbitrary. Any concurrent reachability game with N positions and at most $m \geq 2$ actions in each position has an ϵ-optimal stationary strategy of patience at most $(1/\epsilon)^{m^{O(N)}}$.*

This lemma is an asymptotic improvement of Theorem 4 of Hansen *et al.* [12], that gave an upper bound of $(1/\epsilon)^{2^{30M}}$, for a *total* number of M actions, when $M \geq 10$ and $0 < \epsilon < \frac{1}{2}$. This result does however have the advantage of an *explicit* constant in the exponent, which the bound of Lemma 4 lacks.

Combining Lemma 3, Lemma 4, and also applying inequality (1), we get the following upper bound:

Theorem 5. *Let $\epsilon > 0$ be arbitrary. When applying value iteration or strategy iteration to a concurrent reachability game with N non-terminal positions and $m \geq 2$ choices for each player in each position, after at most $(1/\epsilon)^{m^{O(N)}}$ iterations, an ϵ-approximation to the value has been obtained.*

Also, Lemma 3 will be very useful for us below when applied in the contrapositive. Specifically, below, we will directly analyze and compare the value of $P(N,m)$ with the value of its time bounded version, and use this to conclude that the value iteration algorithm does not converge quickly when applied to this game. The lemma then implies that the patience of any ϵ-optimal strategy is large. When we later consider the strategy iteration algorithm applied to the same game, we will show that the strategy computed after any sub-astronomical number of iterations has too low patience to be ϵ-optimal.

2.2 The Value of Time Bounded Generalized Purgatory and the Complexity of Value Iteration

In this section we give an upper bound on the value of a time bounded version of the Generalized Purgatory game $P(N, m)$. As explained in Section 2.1, this upper bound immediately implies a lower bound on the number of iterations needed by value iteration to approximate the value of the original game.

We let $P_T(N, m)$ be the time bounded version of $P(N, m)$ as defined in Section 2.1, i.e. $P_T(N, m)$ is syntactic sugar for $(P(N, m))_T$. Also, we need to fix an indexing of the positions of $P(N, m)$. We define position i for $i = 1, \ldots, N$ to be the position where Dante already guessed correctly $i - 1$ times in a row and still needs to guess correctly $N - i + 1$ times in a row to win the game.

First we give a rather precise analysis of the one-position case. Besides being interesting in its own right (to establish that value iteration is exponential even for this case), this will also be useful later when we analyze strategy iteration.

Theorem 6. *Let* $m \geq 2$ *and* $T \geq 1$. *The value of position* $(1, T)$ *of* $P_T(1, m)$ *is less than*

$$1 - (1 - \frac{1}{m})(\frac{1}{mT})^{1/(m-1)}.$$

Proof. Let $\epsilon = (1/mT)^{1/(m-1)}$. Consider any strategy (not necessarily stationary) for Dante for playing $P_T(1, m)$. In each round of play, Dante chooses his action with a probability distribution that may depend on previous play and time left. We define a reply by Lucifer in a round-to-round fashion.

Fix a history of play leading to some current round and let $p_1, p_2, \ldots, p_m$ be the probabilities by which Dante plays $1, 2, \ldots, m$ in this current round. There are two cases.

1. There is an i so that $p_i < (\frac{1-\epsilon}{\epsilon}) \sum_{j \geq i+1} p_j$. We call such a round a *green* round. In this case, Lucifer plays i.
2. For all i, $p_i \geq (\frac{1-\epsilon}{\epsilon}) \sum_{j \geq i+1} p_j$. We call such a round a *red* round. In this case, Lucifer plays m.

This completes the definition of Lucifer's reply.

We now analyze the probability that Dante wins $P_T(1, m)$ when he plays his strategy and Lucifer plays this reply. We show this probability to be at most

$$1 - (1 - \frac{1}{m})(\frac{1}{mT})^{1/(m-1)}$$

and we shall be done.

Let us consider a green round. We claim that the probability that Dante wins in this round, conditioned on the previous history of play, and conditioned on play ending in this round, is at most $1 - \epsilon$. Indeed, this conditional probability is given by

$$\begin{aligned}\frac{p_i}{p_i + (p_{i+1} + \cdots + p_m)} &< \frac{(\frac{1-\epsilon}{\epsilon})(\sum_{j \geq i+1} p_j)}{(\frac{1-\epsilon}{\epsilon})(\sum_{j \geq i+1} p_j) + (\sum_{j \geq i+1} p_j)} \\ &= \frac{(1-\epsilon)/\epsilon}{(1-\epsilon)/\epsilon + \epsilon/\epsilon} \\ &= 1 - \epsilon.\end{aligned}$$

Let us next consider a red round. We claim that the probability of play ending in this round, conditioned on the previous history of play, is at most ϵ^{m-1}. Indeed, note that this conditional probability is exactly p_m, and that

$$1 = \sum_{j=1}^{m} p_j = p_1 + \sum_{j=2}^{m} p_j \geq (1 + \frac{1-\epsilon}{\epsilon})(\sum_{j=2}^{m} p_j) = (1 + \frac{1-\epsilon}{\epsilon})(p_2 + \sum_{j=3}^{m} p_j)$$

$$\geq (1 + \frac{1-\epsilon}{\epsilon})^2(\sum_{j=3}^{m} p_j) \geq \cdots \geq (1 + \frac{1-\epsilon}{\epsilon})^{m-1} p_m = (\frac{1}{\epsilon})^{m-1} p_m$$

from which $p_m \leq \epsilon^{m-1}$. That is, in every round of play, conditioned on previous play, either it is the case that the probability that play ends in this round is at most ϵ^{m-1} (for the case of a red round) or it is the case that conditioned on play ending, the probability of win for Dante is at most $1 - \epsilon$ (for the case of a green round).

Now let us estimate the probability of a win for Dante in the entire game $P_T(1, m)$. Let W denote the event that Dante wins. Let G be the event that play ends in a green round. Also, let R be the event that play ends in a red round. Then, we have

$$\begin{aligned}
\Pr[W] &= \Pr[W|R]\Pr[R] + \Pr[W|G]\Pr[G] \\
&\leq \Pr[R] + \Pr[W|G]\Pr[G] \\
&= \Pr[R] + \Pr[W|G](1 - \Pr[R]) \\
&= \Pr[R] + \Pr[W|G] - \Pr[R]\Pr[W|G] \\
&< (\epsilon^{m-1})T + (1 - \epsilon) - (\epsilon^{m-1})T(1 - \epsilon) \\
&= 1 - \epsilon + T\epsilon^m \\
&= 1 - (\frac{1}{mT})^{1/(m-1)} + T(\frac{1}{mT})^{\frac{m}{m-1}} \\
&= 1 - (1 - \frac{1}{m})(\frac{1}{mT})^{1/(m-1)}.
\end{aligned}$$

Combining Lemma 1 with Theorem 6 we get the result that value iteration needs exponential time, even for one-position games.

Corollary 7. *Let $0 < \epsilon < 1$. Applying less than $\frac{1}{em}(1/\epsilon)^{m-1}$ iterations of the value iteration algorithm to $P(1, m)$ yields a valuation at least ϵ smaller than the exact value.*

Next, we analyze the N-position case, where we give a somewhat coarser bound.

Theorem 8. *Let N, m, k, T be integers with $N \geq 2, m \geq 2, 1 \leq k \leq N - 2$ and $T \leq 2^{m^{N-k}}$. Then, the value of $P_T(N, m)$ is at most $2m^{-k} + 2^{-m^{N-k-1}}$.*

Proof. We show an upper bound on the value of $P_T(N, m)$ of $2m^{-k} + 2^{-m^{N-k-1}}$ by exhibiting a particular strategy of Lucifer and showing that any response by Dante to this particular strategy of Lucifer will make Dante win with probability at most $2m^{-k} + 2^{-m^{N-k-1}}$.

To structure the proof, we divide the play into *epochs*. An epoch begins and another ends immediately after each time Dante has guessed incorrectly by undershooting, so that he now finds himself in exactly the same situation as when the play begins (but in general with less time left to win). That is, Dante wins if and only if there is an epoch of length N containing only correct guesses. For convenience, we make the game a little more attractive for Dante by continuing play for T epochs, rather than T rounds. Call this prolonged game G'_T. Clearly, the value of G_T is at most the value of G'_T, so it is okay to prove the upper bound for the latter. We index the epochs $1, 2, \ldots, T$.

To define the strategy of Lucifer, we first define a function $f : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ as follows:

$$f(i,j) = 1 + (j-1)\sum_{r=0}^{i-1} m^r.$$

Then, it is easy to see that f satisfies the following two equations.

$$f(i,m) = m^i \tag{2}$$

$$f(i,j+1) = f(i,j) + \sum_{r=0}^{i-1} f(r,m) \tag{3}$$

The specific strategy of Lucifer is this: Let d be the number of rounds already played in the current epoch. If $d \geq N-k$, Lucifer chooses a number between 1 and m uniformly at random. If $d < N-k$, he hides the numbers $j = 1, \ldots, m-1$ with probabilities $p_j(d) = 2^{-f(N-k-d,m+1-j)}$ and puts all remaining probability mass on the number m (since $N-k-d \geq 1$ and $m \geq 2$, there is indeed some probability mass left for m).

Freeze the strategy of Lucifer to this strategy. From the point of view of Dante, the game G_T is now a finite horizon absorbing Markov decision process. Thus, he has an optimal policy that is deterministic and history independent. That is, the choices of Dante according to this policy depend only on the number of rounds already played in the present epoch and the remaining number of epochs before the limit of T epochs has been played, or, equivalently, on the index of the current epoch. We can assume without loss of generality that Dante plays such an optimal policy. That is, his optimal policy for epoch t can be described by a specific sequence of actions $a_{t0}, a_{t1}, a_{t2}, \ldots, a_{t(N-1)}$ in $\{1, \ldots, m\}$ to make in the next N rounds (with the caveat that this sequence of choices will be aborted if the epoch ends).

Se define the following mutually exclusive events W_t, L_t:

– W_t: Dante wins the game in epoch t (by guessing correctly N times).
– L_t: Dante loses the game in epoch t (by overshooting Lucifer's number)

We make the following claim:

Claim: For each t, either $\Pr[W_t] \leq 2^{-m^{N-k}-m^{N-k-1}}$ or $\Pr[W_t]/\Pr[L_t] \leq 2m^{-k}$.

First, let us see that the claim implies the lemma. Indeed, the probability of Dante winning can be split into the contributions from those epochs where Dante wins with probability at most $2^{-m^{N-k}-m^{N-k-1}}$ and the remaining epochs. The total winning probability mass from the first is at most $T2^{-m^{N-k}-m^{N-k-1}} \leq 2^{-m^{N-k-1}}$ and the total winning probability mass of the rest is at most $2m^{-k}$, giving an upper bound for Dante's winning probability of $2m^{-k} + 2^{-m^{N-k-1}}$.

So let us prove the claim. Fix an epoch t and let $a_{t0}, a_{t1}, a_{t3}, \ldots, a_{t(N-1)}$ be Dante's sequence of actions. Suppose $a_{t0} = 1$ and $a_{t1} = 1$. Then, since Lucifer only plays 1 in the first two rounds with probability $p_1(0)p_1(1) = 2^{-f(N-k,m)} \cdot 2^{-f(N-k-1,m)}$, Dante only wins the game in this epoch with at most that probability, which by equation (2) is equal to $2^{-m^{N-k}-m^{N-k-1}}$, as desired.

Now assume $a_{t0} > 1$ or $a_{t1} > 1$. We want to show that $\Pr[W_t]/\Pr[L_t] \leq 2m^{-k}$. Let d be the largest index so that $d < N-k$ and so that $a_{td} > 1$. Since $a_{t0} > 1$ or $a_{t1} > 1$, such a d exists. Let E be the event that epoch t lasts for at least d rounds. We will show that $\Pr[W_t|E]/\Pr[L_t|E] \leq 2m^{-k}$. Since $W_t \subseteq E$, this also implies that $\Pr[W_t]/\Pr[L_t] \leq 2m^{-k}$. Since we condition on E we look at Dante's decision after d rounds of epoch t. He chooses the action $j = a_{td} > 1$. If Lucifer at this point chooses a number small than j, Dante loses. In particular, since Lucifer chooses the number $j-1$ with probability $2^{-f(N-k-d,m+1-(j-1)}$, Dante loses the entire game by his action a_{td} with probability at least $2^{-f(N-k-d,m-j)}$, conditioned on E. On the other hand the probability that he wins the game in this epoch conditioned on E is at most $(2^{-f(N-k-d,m+1-j)})(\prod_{i=d+1}^{N-k-1} 2^{-f(N-k-i,m)})(m^{-k}))$, the first factor being the probability that Lucifer chooses j at round d, the second factor being the probability that Lucifer like Dante repeatedly chooses 1 until the last k rounds of the epoch begin, and the third factor being the probability that Lucifer matches Dante's choices in those k rounds. Now we have

$$\begin{aligned}
\Pr[W_t]/\Pr[L_t] &\leq \\
\Pr[W_t|E]/\Pr[L_t|E] &\leq \\
(2^{-f(N-k-d,m+1-j)})(\prod_{i=d+1}^{N-k-1} 2^{-f(N-k-i,m)})(m^{-k}))2^{f(N-k-d,m-j)} &\leq \\
m^{-k}2^{f(N-k-d,m-j)-f(N-k-d,m+1-j)-\sum_{r=1}^{N-k-d-1} f(r,m)} &= \\
2m^{-k}2^{f(N-k-d,m-j)-f(N-k-d,m+1-j)-\sum_{r=0}^{N-k-d-1} f(r,m)} &= \\
2m^{-k}
\end{aligned}$$

as desired.

Combining Lemma 1 with Theorem 8 we get the result that value iteration needs doubly exponential time to obtain any non-trivial approximation:

Corollary 9. *Let N be even. Applying less than $2^{m^{N/2}}$ iterations of the value iteration algorithm to $P(N,m)$ yields a valuation of the initial position of at most $3m^{-N/2}$, even though the actual value of the game is 1.*

We also get the following bound on the patience of near-optimal strategies of $P(N,m)$ that will be useful when analyzing strategy iteration.

Theorem 10. *Suppose N is sufficiently large and $m \geq 2$. Let $\epsilon = 1 - 4m^{-N/2}$. Then all ϵ-optimal strategies of $P(N,m)$ have patience at least $2^{m^{N/3}}$.*

Proof. Putting $c = \frac{N \ln m}{2}$, Lemma 2 tells us that if $P(N,m)$ has an ϵ-optimal strategy of patience less than $l = 2^{m^{N/3}}$, then the value of $P_t(N,m)$ is at least $1 - \epsilon - e^{-c} = 3m^{-N/2}$, where $t = cNl^N \le 2^{m^{N/2}}$. But putting $k = N/2$, Theorem 8 tells us that the value of $P_t(N,m)$ is at most $2m^{-N/2} + 2^{-m^{N/2-1}} < 3m^{-N/2}$, a contradiction.

2.3 Strategy Iteration

The technical content of this section is a number of lemmas on what happens when the strategy iteration algorithm is applied to $P(N,m)$, leading up to the following crucial lemma:

Lemma 11. *When applying strategy iteration to $P(N,m)$, the patience of the strategy x^t computed in iteration t is at most $e \cdot m \cdot t$.*

Before we prove Lemma 11, we show that it implies the lower bound we are looking for.

Theorem 12. *Suppose N is sufficiently large. Applying less than $2^{m^{N/4}}$ iterations of strategy iteration to $P(N,m)$ yields a valuation of the initial position of less than $4m^{-N/2}$, despite the fact that the value of the position is* 1.

Proof. Lemma 11 implies that the patience of the strategy x^t computed in iteration t for $t = 2^{m^{N/4}}$ is at most $em2^{m^{N/4}}$. Theorem 10 states that if $\epsilon = 1 - 4m^{-N/2}$, then all ϵ-optimal strategies of $P(N,m)$ have patience at least $2^{m^{N/3}}$. So x^t is not ϵ-optimal and the bound follows.

Due to space constraints, the rather long and technical proof of Lemma 11 itself is omitted, but can be found in the full version of this paper [10].

Acknowledgement

First and foremost, we would like to thank Uri Zwick for extremely helpful discussions and Kousha Etessami for being instrumental for starting this research. We would also like to thank Vladimir V. Podolskii for helpful discussions.

References

1. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: Strategy improvement for concurrent reachability games. In: Third International Conference on the Quantitative Evaluation of Systems, pp. 291–300. IEEE Press, New York (2006)
2. Chatterjee, K., de Alfaro, L., Henzinger, T.A.: Termination criteria for solving concurrent safety and reachability games. In: 20th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 197–206. SIAM, Philadelphia (2009)
3. Chatterjee, K., Majumdar, R., Jurdziński, M.: On nash equilibria in stochastic games. In: Marcinkowski, J., Tarlecki, A. (eds.) CSL 2004. LNCS, vol. 3210, pp. 26–40. Springer, Heidelberg (2004)

4. Condon, A.: On algorithms for simple stochastic games. In: Advances in Computational Complexity Theory, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 13, pp. 51–73 (1993)
5. Dai, D., Ge, R.: New results on simple stochastic games. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 1014–1023. Springer, Heidelberg (2009)
6. de Alfaro, L., Henzinger, T.A., Kupferman, O.: Concurrent reachability games. Theor. Comput. Sci. 386, 188–217 (2007)
7. Etessami, K., Yannakakis, M.: Recursive concurrent stochastic games. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 324–335. Springer, Heidelberg (2006)
8. Everett, H.: Recursive games. In: Kuhn, H.W., Tucker, A.W. (eds.) Contributions to the Theory of Games Vol. III. Annals of Mathematical Studies, vol. 39, pp. 47–78. Princeton University Press, Princeton (1957)
9. Friedmann, O.: An exponential lower bound for the parity game strategy improvement algorithm as we know it. In: 24th Annual IEEE Symposium on Logic in Computer Science, pp. 145–156. IEEE Press, New York (2009)
10. Hansen, K.A., Ibsen-Jensen, R., Miltersen, P.B.: The complexity of solving reachability games using value and strategy iteration, http://arxiv.org/abs/1007.1812
11. Hansen, K.A., Koucký, M., Lauritzen, N., Miltersen, P.B., Tsigaridas, E.: Exact Algorithms for Solving Stochastic Games. In: 43rd ACM Symposium on Theory of Computing, ACM Press, New York (2011)
12. Hansen, K.A., Koucký, M., Miltersen, P.B.: Winning concurrent reachability games requires doubly exponential patience. In: 24th Annual IEEE Symposium on Logic in Computer Science, pp. 332–341. IEEE Press, New York (2009)
13. Himmelberg, C.J., Parthasarathy, T., Raghavan, T.E.S., Vleck, F.S.V.: Existence of p-equilibrium and optimal stationary strategies in stochastic games. Proc. Amer. Math. Soc. 60, 245–251 (1976)
14. Hoffman, A., Karp, R.: On nonterminating stochastic games. Management Science 12, 359–370 (1966)
15. Howard, R.: Dynamic Programming and Markov Processes. MIT Press, Cambridge (1960)
16. Mertens, J.F., Neyman, A.: Stochastic games. International Journal of Game Theory 10, 53–66 (1981)
17. Parthasarathy, T.: Discounted and positive stochastic games. Bull. Amer. Math. Soc. 77, 134–136 (1971)
18. Rao, S., Chandrasekaran, R., Nair, K.: Algorithms for discounted games. J. Optimiz. Theory App. 11, 627–637 (1973)
19. Shapley, L.S.: Stochastic games. Proceedings of the National Academy of Sciences, U.S.A. 39, 1095–1100 (1953)

Faster Polynomial Multiplication via Discrete Fourier Transforms

Alexey Pospelov[⋆]

Saarland University, Computer Science Department
pospelov@cs.uni-saarland.de

Abstract. We study the complexity of polynomial multiplication over arbitrary fields. We present a unified approach that generalizes all known asymptotically fastest algorithms for this problem and obtain faster algorithms for polynomial multiplication over certain fields which do not support DFTs of large smooth orders. We prove that the famous Schönhage-Strassen's upper bound cannot be improved over the field of rational numbers if we consider only algorithms based on consecutive applications of DFT, as all known fastest algorithms are.

This work is inspired by the recent improvement for the closely related problem of complexity of integer multiplication by Fürer and its consequent modular arithmetic treatment due to De, Kurur et al. We explore the barriers in transferring the techniques for solutions of one problem to a solution of the other.

Keywords: Polynomial multiplication, discrete Fourier transform, algebraic complexity, lower bounds.

1 Introduction

Complexity of polynomial multiplication is one of the central problems in computer algebra and algebraic complexity theory. Given two univariate polynomials by vectors of their coefficients,

$$a(x) = \sum_{i=0}^{n-1} a_i x^i\,, \qquad\qquad b(x) = \sum_{j=0}^{n-1} b_j x^j\,, \tag{1}$$

over some field k, the goal is to compute the coefficients of their product

$$c(x) = a(x)\cdot b(x) = \sum_{\ell=0}^{2n-2} c_\ell x^\ell = \sum_{\ell=0}^{2n-2} \sum_{\substack{0\le i,\,j<n,\\ i+j=\ell}} a_i b_j x^\ell\,. \tag{2}$$

The direct way by the formulas above requires n^2 multiplications and $(n-1)^2$ additions of elements of k, making the total complexity of the naive algorithm $O(n^2)$. In what follows we call k the *ground field*.

[⋆] This research is supported by Cluster of Excellence "Multimodal Computing and Interaction" at Saarland University.

1.1 Model of Computation

We study the problem of the total *algebraic* complexity of the multiplication of polynomials over *fields*. That is, elements of k are thought of as algebraic entities, and each binary arithmetic operation on these entities has unit cost. This model is rather abstract in the sense that it counts, for example, an infinite precision multiplication of two reals as a unit cost operation. On the other hand, it has an advantage of being independent of any concrete implementation that may depend on many factors, including human-related, thus it is more universal, see the discussion on this topic in [3, Introduction].

We are concerned with the *total* number of arithmetic operations, *i.e.,* multiplications and additions/subtractions that are sufficient to multiply two degree $n-1$ polynomials. Since the resulting functions can be computed without divisions, it seems natural to consider only *division-free algebraic algorithms.* The inputs of such algorithms are the values $a_0, \ldots, a_{n-1}, b_0, \ldots, b_{n-1} \in k$, the outputs are the values $c_0, c_1, \ldots, c_{2n-2} \in k$ as defined in (1), (2). Any step of an algorithm is a multiplication, an addition or a subtraction of two values, each being an input, a value previously computed by the algorithm, or a constant from the ground field. An algorithm computes the product of two degree $n-1$ polynomials if all outputs $c_0, \ldots, c_{2n-2}$ are computed in some of its steps. The number of steps of an algorithm $\mathcal{A}$ is called *algebraic* or *arithmetic* complexity of $\mathcal{A}$. Note that a division-free algebraic algorithm is a special instance of the well-known straight-line program model. In what follows, if not mentioned explicitly we will always consider division-free algebraic algorithms.

A multiplication performed in a step of an algorithm is called *scalar* if at least one multiplicand is a field constant, and *nonscalar* in the other case. For an algorithm $\mathcal{A}$ which computes the product of two degree $n-1$ polynomials, we define $L^m_{\mathcal{A}}(n)$ to be the number of nonscalar multiplications used in $\mathcal{A}$, and $L^a_{\mathcal{A}}(n)$ to be the total number of additions, subtractions and scalar multiplications in $\mathcal{A}$. We also set $L_{\mathcal{A}}(n) \coloneqq L^m_{\mathcal{A}}(n) + L^a_{\mathcal{A}}(n)$, the *total algebraic complexity* of $\mathcal{A}$ computing the product of two degree $n-1$ polynomials. In what follows, $\mathbf{A}^n_k$ always stands for the set of division-free algorithms computing the product of two degree $n-1$ polynomials over k, $L^m_k(n) \coloneqq \min_{\mathcal{A}\in\mathbf{A}^n_k} L^m_{\mathcal{A}}(n)$, $L^a_k(n) \coloneqq \min_{\mathcal{A}\in\mathbf{A}^n_k} L^a_{\mathcal{A}}(n)$, $L_k(n) \coloneqq \min_{\mathcal{A}\in\mathbf{A}^n_k} L_{\mathcal{A}}(n)$. When the field k will be clear from the context or insignificant, we will use the simplified notation: $L^m(n)$, $L^a(n)$ and $L(n)$, respectively. Note that $L(n)$ need not be equal to $L^m(n) + L^a(n)$, since the minimal number of nonscalar multiplications and the minimal number of additive operations and scalar multiplications can be achieved by different algorithms.

It is known [18], [3, Section 14.1] that an algebraic algorithm $\mathcal{A}_d$ with divisions for a set of l quadratic forms of n variables over an infinite field k can be transformed into a division-free algorithm $\mathcal{A}$ such that $L^m_{\mathcal{A}}(n)$ will not exceed the total number of nonscalar multiplications and divisions used by $\mathcal{A}_d$ for computing the original l quadratic forms of n variables. It is unknown, if the same result holds also over finite fields. It is also an open problem how the total complexity of a set of quadratic forms is affected by allowing divisions. Strassen proved in [18] that the *exponent* of matrix multiplication is the same in the division-free model and in the model with divisions. We mention here however, that the notion of the exponent is very rough and does not even define the order of the complexity: two algorithms $\mathcal{A}$ and $\mathcal{A}'$ with $L_{\mathcal{A}}(n) = \Theta(n)$ and $L_{\mathcal{A}'}(n) = \Theta(n \log^2 n)$ both have the complexity exponent 1, while the complexity of $\mathcal{A}'$ is obviously higher.

1.2 Fast Polynomial Multiplication and Lower Bounds

Designing efficient algorithms and proving lower bounds is a classical problem in algebraic complexity theory that received wide attention in the past. For an exhaustive treatment of the current state of the art we advise the reader to refer to [3, Sections 2.1, 2.2, 2,7, 2.8], [11, Chapter 8]. There exists an algorithm $\mathcal{A} \in \mathbf{A}_k^n$, such that

$$L_{\mathcal{A}}^m(n) = O(n)\,, \qquad L_{\mathcal{A}}^a(n) = O(n \log n)\,, \qquad L_{\mathcal{A}}(n) = O(n \log n)\,,^{1} \tag{3}$$

if k supports Discrete Fourier Transformation (DFT) of order 2^l, [3, Chapter 1, Section 2.1] or 3^l, [3, Exercise 2.5] for each $l > 0$. Schönhage-Strassen's algorithm $\mathcal{B} \in \mathbf{A}_k^n$ computes the product of two degree $n-1$ polynomials over an arbitrary field k of characteristic different from 2 with

$$\begin{gathered} L_{\mathcal{B}}^m(n) = O(n \log n)\,, \qquad L_{\mathcal{B}}^a(n) = O(n \log n \log\log n)\,, \\ L_{\mathcal{B}}(n) = O(n \log n \log\log n)\,. \end{gathered} \tag{4}$$

cf. [16], [3, Section 2.2], [11, Sections 8.2, 8.3]. In fact, the original algorithm of [16] computes the product of two n-bit integers, but it readily transforms into an algorithm for degree $n-1$ polynomial multiplication. For fields of characteristic 2, Schönhage's algorithm [15], [3, Exercise 2.6] has the same upper bounds as in (4). An algorithm $\mathcal{C}'$ for multiplication of polynomials over arbitrary rings with the same upper bound for $L_{\mathcal{C}'}^m(n)$ was first proposed by Kaminski in [12]. However, there was no matching upper bound for $L_{\mathcal{C}'}^a(n)$. Cantor and Kaltofen generalized Schönhage-Strassen's algorithm into an algorithm $\mathcal{C}$ for the problem of multiplication of polynomials over arbitrary algebras (not necessarily commutative, not necessarily associative) achieving the upper bounds (4), see [5].

For the rest of the paper, we will use the introduced notation: $\mathcal{A}$ will always stand for the multiplication algorithm via DFT with complexity upper bounds (3), $\mathcal{B}$ will stand for Schönhage-Strassen's algorithm if $\operatorname{char} k \neq 2$ and for Schönhage's algorithm if $\operatorname{char} k = 2$, both with complexity upper bounds (4), and $\mathcal{C}$ will stand for Cantor-Kaltofen's algorithm for multiplication of polynomials over arbitrary algebras with the same complexity upper bounds as Schönhage-Strassen's algorithm.

Upper and lower bounds for $L_k^m(n)$, which is also called the *multiplicative complexity*, received special attention in the literature, see, e.g., [3, Section 14.5]. It is interesting, that for each k, there exists always an algorithm $\mathcal{E} \in \mathbf{A}_k^n$ with $L_{\mathcal{E}}^m(n) = O(n)$, if we do not worry that $L_{\mathcal{E}}^a(n)$ will be worse than in (4), see [6,17].

[1] In this paper we always use $\log := \log_2$.

There are few lower bounds for the algebraic complexity of polynomial multiplication. Most of them are actually bounding $L^m(n)$ which can be used as a conservative lower bound for $L(n)$. Since the coefficients $c_0, \ldots, c_{2n-2}$ are linearly independent, in case of division-free algorithms one immediately obtains the lower bound

$$L(n) \geq L^m(n) \geq 2n - 1$$

over arbitrary fields. Currently, this is the only general lower bound for $L(n)$ which does not depend on the ground field. Bürgisser and Lotz in [4] proved the only currently known nonlinear lower bound of $\Omega(n \log n)$ for $L_{\mathbb{C}}(n)$ (actually, on $L^a_{\mathbb{C}}(n)$) which holds in case when all scalar multiplications in an algorithm are with bounded constants.

The gap between the upper and the lower bounds on $L_k(n)$ motivates to look for better multiplication algorithms and for higher lower bounds for the complexity of polynomial multiplication, in particular over small fields. For example, it is still an open problem if the total algebraic complexity of polynomial multiplication is nonlinear, see [3, Problem 2.1]. Another well known challenge is to decrease the upper bound for $L_k(n)$ of (4) to the level of (3) in case of arbitrary fields, see [13] for the more general question on multivariate polynomial multiplication. In this paper we address both these problems.

1.3 Our Results

As our first contribution, for every field k, we present an algorithm $\mathcal{D}_k \in \mathbf{A}^n_k$, which is a generalization of Schönhage-Strassen's construction that works over arbitrary fields and achieves the best known complexity upper bounds. In fact, we argue that the algorithm $\mathcal{D}_k$ stands for a generic polynomial multiplication algorithm that relies on consecutive application of DFT. In particular, the algorithms $\mathcal{A}$, $\mathcal{B}$, and $\mathcal{C}$ come as special cases of the algorithm $\mathcal{D}_k$. We are currently not aware of any algorithms with an upper bound of (4) that are not based on consecutive DFT applications and thus do not follow from the algorithm $\mathcal{D}_k$.

As the second contribution, we show that $L_{\mathcal{D}_k}(n) = o(n \log n \log \log n)$ in case when algorithm $\mathcal{A}$ cannot be applied but the field k has some simple algebraic properties that are ignored by algorithms $\mathcal{B}$ and $\mathcal{C}$. This improves the upper bound of (4) over such fields. We also present a parameterization of fields k with respect to the performance of the algorithm $\mathcal{D}_k$, and give explicit upper bounds which depend on this parameterization. More precisely, over each field k, we have $\Omega(n \log n) = L_{\mathcal{D}_k}(n) = O(n \log n \log \log n)$, and over certain fields that do not admit low-overhead application of the algorithm $\mathcal{A}$, the algorithm $\mathcal{D}_k$ achieves intermediate complexities between the indicated bounds.

Finally, we show, that the algorithm $\mathcal{D}_k$ has natural limitations depending on the ground field k. For example, we prove that $L_{\mathcal{D}_{\mathbb{Q}}}(n) = \Omega(n \log n \log \log n)$. Furthermore, we characterize all such fields where the application of DFT-based methods does not lead to any improvement of the upper bound (4). Therefore, we consider this as an exhaustive exploration of the performance of generic algorithms for polynomial multiplication based on DFT.

1.4 Organization of the Paper

Section 2 contains the necessary algebraic preliminaries. In Section 3 we recall the best known upper bounds for the computation of DFT over different fields and show an efficient application of their combination. We also indicate limitations of the known techniques.

Section 4 contains our main contributions. We end with one particular number-theoretic conjecture due to Bläser on the existence of special finite field extensions. In fact, if it holds then the algorithm $\mathcal{D}_k$ presented in Section 4 has lower algebraic complexity than that of the previously known algorithms $\mathcal{B}$ and $\mathcal{C}$ over any field of characteristic different from 0.

2 Basic Definitions

In what follows we will denote the ground field by k. *Algebra* will always stand for a finite dimensional associative algebra over some field with unity 1. For a function $f : \mathbb{N} \to \mathbb{R}$, a positive integer n is called *f-smooth*, if each prime divisor of n does not exceed $f(n)$. Note that this definition is not trivial only if $f(n) < \frac{n}{2}$. If $f(n) = O(1)$ then an f-smooth positive integer is called just *smooth.*

All currently known fastest algorithms for polynomial multiplication over arbitrary fields rely on the possibility to apply the Discrete Fourier Transform by means of the Fast Fourier Transform algorithm (FFT) and on the estimation of the overhead needed to extend the field to make DFTs available. This possibility depends on existence of so-called *principal roots of unity* of large smooth orders, e.g., of orders 2^ν for all $\nu > 0$.

Let A be an algebra over k. $\omega \in A$ is called a *principal n-th root of unity* if $\omega^n = 1_A$ (where 1_A is the unity of A) and for $1 \le \nu < n$, $1 - \omega^\nu$ is not a zero divisor in A. It follows that if $\omega \in A$ is a principal n-th root of unity then $\operatorname{char} k \nmid n$ and

$$\sum_{\nu=0}^{n-1} \omega^{i\cdot\nu} = \begin{cases} n, & \text{if } i \equiv 0 \pmod{n}, \\ 0, & \text{otherwise}. \end{cases} \tag{5}$$

If A is a field, then $\omega \in A$ is a principal n-th root of unity iff ω is a primitive n-th root of unity. For a principal n-th root of unity $\omega \in A$, the map $\mathrm{DFT}_n^\omega : A[x]/(x^n - 1) \to A^n$ defined as $\mathrm{DFT}_n^\omega \left(\sum_{\nu=0}^{n-1} a_\nu x^\nu\right) = (\tilde{a}_0, \ldots, \tilde{a}_{n-1})$, where $\tilde{a}_i = \sum_{\nu=0}^{n-1} \omega^{i\cdot\nu} a_\nu$, for $i = 0, \ldots, n-1$, is called the *Discrete Fourier Transform* of order n over A with respect to the principal n-th root of unity ω.

It follows from the Chinese Remainder Theorem that if $\omega \in A$ is a principal n-th root of unity, then DFT_n^ω is an isomorphism between $A[x]/(x^n - 1)$ and A^n. (5) implies that the inverse transform of DFT_n^ω can be computed by $\frac{1}{n}\mathrm{DFT}_n^{\omega^{-1}}$ since ω^{-1} is also a principal n-th root of unity in A [3, Theorem (2.6)]: $a_i = \frac{1}{n}\sum_{\nu=0}^{n-1} \omega^{-i\cdot\nu} \cdot \tilde{a}_\nu$, for $i = 0, \ldots, n-1$. Note that if $\omega \in A$ is a principal n-th root of unity and $a(x) = \sum_{\nu=0}^{n-1} a_\nu x^\nu \in k[x]/(x^n - 1)$ then $\mathrm{DFT}_n^\omega(a(x)) = \left(a(\omega^0), \ldots, a(\omega^{n-1})\right)$.

An important property of the DFT is that it can be computed efficiently under certain conditions, see Section 3. We only mention here that if for some constant s there exists a principal s^n-th root of unity ω in an algebra A, then $\mathrm{DFT}^{\omega}_{s^n}$ can be computed in $O(s^n \log s^n)$ additions of elements in A and $O(s^n \log s^n)$ multiplications by powers of ω in A.

3 An Upper Bound on the Complexity of DFT

In this section we summarize the best known upper bounds for the computation of DFTs over an algebra A with unity 1. Let $\omega \in A$ be a principal n-th root of unity. For $a(x) \in A[x]$ of degree $n-1$, let $\tilde{a} = \mathrm{DFT}^{\omega}_{n}(a(x)) \in A^n$. We will denote the total number of operations over A that are sufficient for an algebraic algorithm to compute the DFT of order n over A by $D_A(n)$.

There is always an obvious way to compute $\tilde{a}$ from the coefficients of $a(x)$.

Lemma 1. *For every A such that the DFT of order $n \geq 1$ is defined over A,*

$$D_A(n) \leq \begin{cases} 2n^2 - 3n + 1, & \text{if } 2 \nmid n\,, \\ 2n^2 - 5n + 4, & \text{if } 2 \mid n\,. \end{cases} \tag{6}$$

The next method of effective reduction of a DFT of large order to DFTs of smaller orders is known as Cooley-Tukey's algorithm [8], [7, Section 4.1] and is based on the following lemma which directly follows from well-known facts and is given here for completeness.

Lemma 2. *Let the DFT of order*

$$n = p_1^{d_1} \ldots p_s^{d_s} \geq 2 \tag{7}$$

be defined over A. Then

$$D_A(n) \leq n \sum_{\sigma=1}^{s} \left(\frac{d_\sigma}{p_\sigma} (D_A(p_\sigma) - 1) + d_\sigma \right) - n + 1\,. \tag{8}$$

Corollary 1. *Let n be as in (7), and let all $2 = p_1 < p_2 < \cdots < p_s$ be primes (since $p_1 = 2$, we allow here the case $d_1 = 0$). Then*

$$D_A(n) \leq \left(\frac{3}{2} d_1 + 2 \sum_{\sigma=2}^{s} d_\sigma (p_\sigma - 1) - 1 \right) n + 1\,, \tag{9}$$

$$D_A(n) \leq 2 \max_{1 \leq \sigma \leq s} p_\sigma \cdot n \log n\,. \tag{10}$$

Proof. (9) follows from (8) via the upper bound of Lemma 1 for $D_A(p_\sigma)$. Using the fact that the $n = \prod_{\sigma=1}^{s} p_\sigma^{d_\sigma}$ and $p_\sigma \geq 2$ for $1 \leq \sigma \leq s$ implies $\sum_{\sigma=1}^{s} d_i \leq \log n$, we obtain $D_A(n) \leq \left(\frac{3}{2} + 2\left(\max_{1 \leq \sigma \leq s} p_\sigma\right) - 3\right) n \log n + 1$, which proves (10). □

Lemma 2 provides an efficient method of reduction of a DFT of composite order n to several DFTs of smaller orders which divide n. For example, if all p_σ in (7) are bounded by some constant, then (10) shows that Cooley-Tukey's algorithm computes the DFT of order n in $O(n \log n)$ steps. Furthermore, if $\max_{1 \le \sigma \le s} p_\sigma \le g(n)$ for a slowly growing function $g(n)$, say $g(n) = o(\log \log n)$, then (10) gives an upper bound of $O(n \log n \cdot g(n))$ for the computation of the DFT of order n. However, this method fails to be effective if n has large prime factors (or is just prime). We could use the algorithm from Lemma 1, but sometimes we can apply Rader's or Bluestein's algorithm to compute a DFT of prime order [14], [7, Section 4.2].

Lemma 3. *Assume that the DFT of a prime order p is defined over A.*

1. *If the DFT of order $p-1$ is defined over A, then $D_A(p) \le 2D_A(p-1)+O(p)$.*
2. *If the DFT of order $n > 2p-4$ is defined over A, then $D_A(p) \le 2D_A(n)+O(n)$.*

Remark 1. Note that the first bound can be efficient if $p-1$ is a smooth number. Otherwise we may choose some larger smooth n for the second case, making sure that the DFT of order n exists over A and n is not too large, in order to achieve a $O(p \log p)$ upper bound for $D(p)$.

Corollary 2. *Let p be a fixed odd prime, k be a field where the DFT of order $p^N - 1$ is defined for $N = 2^n$, $n \ge \lceil \log(2p-5) \rceil$. Then $D_k(p^N - 1) = O(p^N \cdot N^2)$.*

Proof. We have $p^N - 1 = (p-1)(p+1)(p^2+1)\cdots(p^{2^{n-1}}+1)$. Since p is odd, each factor is even and $p^N - 1 = 2^n \cdot \frac{p-1}{2} \prod_{i=1}^{n-1} \frac{p^{2^i}+1}{2}$. Let $p^N - 1 = p_1^{d_1} p_2^{d_2} \cdots p_s^{d_s}$ be the prime decomposition of $p^N - 1$, so that $2 = p_1 < p_2 < \cdots < p_s$. Let $i_1 \le i_2 \le \cdots$ be such that $p_2, \ldots, p_{i_1} \le \frac{p-1}{2}$, $p_{i_1+1}, \ldots, p_{i_2} \le \frac{p+1}{2}$, etc., and $p_{i_j+1}, \ldots, p_{i_{j+1}} \le \frac{p^{2^{j-1}}+1}{2}$. Note that $i_{n'} = s$ for some $n' \le n$. We also set $i_{-1} = 0$, $i_0 = 1$. From (8) we have

$$D(p^N - 1) \le (p^N - 1) \sum_{\sigma=1}^{s} \left(\frac{d_\sigma}{p_\sigma}(D(p_\sigma) - 1) + d_\sigma \right) - p^N + 2\,.$$

Obviously, for $p_1 = 2$, we have $D(p_1) = 2 \le p_1 \cdot \log p_1$. Using Lemma 3 we can compute the DFT of orders p_σ for $p_\sigma = 2, \ldots, i_2$ in $8p_\sigma \log p_\sigma + O(p_\sigma)$ time since we can reduce each DFT of order p_σ to 2 DFTs of order $2^{n_1} > 2p_\sigma - 4$, and $2^{n_1} < 4p_\sigma$. This is possible since the DFT of order $2^n > 2 \cdot \frac{p-1}{2} - 4$ is defined over k. In the same way, the DFT of order p_σ for $\sigma = i_1 + 1, \ldots, i_2$ can be computed in $16 p_\sigma \log p_\sigma + O(p_\sigma)$ steps since $2^n \cdot \frac{p-1}{2} > 2 \cdot \frac{p+1}{2} - 4$. Continuing this process we obtain the following upper bound:

$$\begin{aligned} D(p^N - 1) &\le (p^N - 1) \sum_{j=-1}^{n-1} \sum_{\sigma=i_j+1}^{i_{j+1}} O\left(d_\sigma \cdot 2^j \log p_\sigma + d_\sigma\right) \\ &= O(p^N \cdot N \cdot \log \prod_{\sigma=1}^{s} p_\sigma^{d_\sigma}) = O(p^N \cdot N^2)\,. \end{aligned}$$ □

Remark 2. For a fixed odd prime p, the DFT of order $p^{2^n}-1$ is defined in the field $\mathbb{F}_{p^{2^n}}$ since the multiplicative group $\mathbb{F}^*_{p^{2^n}}$ of order $p^{2^n}-1$ is cyclic. Corollary 2 implies that the DFT of order $p^{2^n}-1$ can be computed in $O(p^{2^n}\cdot 2^{2n})$ steps over $\mathbb{F}_p$. A similar argument shows that the same holds for any field of characteristic p which contains $\mathbb{F}_{p^{2^n}}$ as a subfield.

4 Unified Approach for Fast Polynomial Multiplication

In this section we present our main contribution. We proceed as follows: first we introduce the notions of the degree function and of the order sequence of a field. Then we describe the DFT-based algorithm $\mathcal{D}_k$ which computes the product of two polynomials over a field k. We show that $\mathcal{D}_k$ generalizes any algorithm for polynomial multiplication that relies on consecutive applications of DFT, and in particular, Schönhage-Strassen's [16], Schönhage's [15], and Cantor-Kaltofen's [5] algorithms for polynomial multiplication are special cases of the algorithm $\mathcal{D}_k$. We prove that both the upper and the lower bounds for the total complexity of the algorithm $\mathcal{D}_k$ depend on the degree function of k and the existence of special order sequences for k. In particular, we show that $L_{\mathcal{D}_k}(n)=\Omega(n\log n)$ when k is a finite field, and $L_{\mathcal{D}_\mathbb{Q}}(n)=\Omega(n\log n\log\log n)$. Furthermore, we show sufficient conditions on the field k for the algorithm $\mathcal{D}_k$ to compute the product of two degree n polynomials in $o(n\log n\log\log n)$ operations over k, that is, to outperform Schönhage-Strassen's, Schönhage's and Cantor-Kaltofen's algorithms. Finally, we pose a number-theoretic conjecture whose validity would imply faster polynomial multiplication over arbitrary fields of positive characteristic.

In what follows k always stands for a field.

4.1 Extension Degree and Order Sequence

Definition 1. *The* degree function *of k is f_k such that $f_k(n)=[k(\omega_n):k]$ for* $\operatorname{char} k\nmid n$*, where ω_n is a primitive n-th root of unity in the algebraic closure of k. If* $\operatorname{char} k \mid n$*, then $f_k(n)$ is* not defined*.*

For example, $f_k(n)=1$ if k is algebraically closed, $f_\mathbb{R}(n)=1$ if $n\le 2$ and $f_\mathbb{R}(n)=2$ for $n\ge 3$, $f_\mathbb{Q}(n)=\phi(n)$ where $\phi(N)$ is the Euler's totient function.

Note that the degree function is well-defined since $[k(\omega_n):k]$ does not depend on the choice of the primitive n-th root of unity ω_n in the algebraic closure of k.

An important idea behind Fürer's algorithm [10,9] is a field extension of small degree containing a principal root of unity of high smooth order. In the case of integer multiplication, the characteristic of the ground ring is a parameter we can choose [9], and it allows us to pick $\mathbb{Z}_{p^c}$ such that p^c-1 has a large smooth factor. However, in the case of multiplication of polynomials over fields, we cannot change the characteristic of the ground field. In what follows we explore this limitation.

Definition 2. *An integer $n > 0$ is called* c-suitable *over the field k if the DFT of order n is defined over k and if $D_k(n) \le cn \log n$. n is called simply* suitable *if it is $O(1)$-suitable.*

It follows from Corollary 1 that any c-smooth n is $2c$-suitable over k as long as the DFT of order n is defined over k, and Lemma 3 also implies that if for each prime divisor p of n, p or $p-1$ or some $n' \ge 2p-3$, $n' = O(p)$, is c-suitable over k, then n is $O(c)$-suitable. If $\operatorname{char} k \ge 3$, then the integers $(\operatorname{char} k)^{2^n} - 1$ are $O(2^n)$-suitable over k for arbitrary n (see Remark 2).

Definition 3. *Let $s(n) : \mathbb{N} \to \mathbb{R}$ be a function such that $s(n) > 1$ for all $n \in \mathbb{N}$. A sequence $\mathcal{N} = \{n_1, n_2, \dots\}$ is called an* order sequence *of sparseness $s(n)$ for the field k, if $n_i < n_{i+1} \le s(n_i)n_i$ and $n_i \mid n_{i+1}$ for $i \ge 1$, and $n_i = n_i' n_i''$ such that there exists a ring extension of k of degree n_i' containing an n_i''-th principal root of unity $\omega_{n_i''}$, so that n_i'' is $O(1)$-suitable over this extension. If $s(n) \le C$ for some constant C, then $\mathcal{N}$ is called an order sequence of* constant sparseness.

It follows from Remark 2 that $n_i = 2^i \cdot (p^{2^i} - 1)$ is almost an order sequence of sparseness $s(n) = n$ for any field of characteristic p. Decreasing the upper bound for the computation of DFT from $O(n \log^2 n)$ to $O(n \log n)$ would turn it into an order sequence of sparseness n.

Remark 3. If $\operatorname{char} k \ne 2$ then, for the order sequence $\mathcal{N} = \{2^i\}_{i \ge 1}$, $f_k(n'') \le \frac{n''}{2}$ for each $n = n'n'' \in \mathcal{N}$, since if for $n \in \mathcal{N}$, $n' := 2^{\lceil \frac{i-1}{2} \rceil} \le n'' := 2^{\lfloor \frac{i-1}{2} \rfloor + 1} \le 2n'$, $\omega_{n''}$ is a primitive n''-th root of unity in the algebraic closure of k, then for some $p(x) \mid x^{\frac{n''}{2}} + 1$, $k(\omega_{n''}) \cong k[x]/p(x)$. The same argument shows that if $\operatorname{char} k \ne 3$ and $\mathcal{N} = \{2 \cdot 3^i\}_{i \ge 1}$, then $f_k(\hat{n}'') \le \frac{2\hat{n}''}{3}$ for each $\hat{n} = \hat{n}'\hat{n}'' \in \mathcal{N}$, $\hat{n}' := 2 \cdot 3^{\lceil \frac{i-1}{2} \rceil}$, $\hat{n}'' := 3^{\lfloor \frac{i-1}{2} \rfloor + 1} \le \frac{3}{2}\hat{n}'$, since for some $p(x) \mid x^{\frac{2\hat{n}''}{3}} + x^{\frac{\hat{n}''}{3}} + 1$, $k(\omega_{\hat{n}''}) \cong k[x]/p(x)$. Both these order sequences have constant sparsenesses.

Definition 4. *A field k is called*

- $t(n)$-Fast, *if there exists an order sequence $\mathcal{N}$ of constant sparseness such that $f_k(n_i'') \le t(n_i'')$ for all $n_i = n_i' n_i'' \in \mathcal{N}$.*
- $t(n)$-Slow, *if for any order sequence $\mathcal{N}$ of constant sparseness, $f_k(n_i'') \ge t(n_i'')$ for all $n_i = n_i' n_i'' \in \mathcal{N}$.*

An $O(1)$-fast field is called just fast.

For example, any algebraically closed field is fast, $\mathbb{R}$ is a fast field, and $\mathbb{Q}$ is a $\phi(n)$-slow field. It follows, that $\mathbb{Q}$ is an $\Omega(\frac{n}{\log \log n})$-slow field [1, Theorem 8.8.7]. Remark 3 implies that any field of characteristic different from 2 is $\frac{n}{2}$-fast, and any field of characteristic different from 3 is $\frac{2n}{3}$-fast.

If we want to extend a $b(n)$-slow field k with an n-th root of unity, the degree of the extension will be $\Omega(b(n))$. In a DFT-based algorithm we need to take an extension $K \supseteq k$ of degree n_1 over k such that K contains a principal n_2-th root of unity. We will see that to increase performance of computing the product of two degree $n-1$ polynomials over k we will want n_2 to be a large suitable number and to belong to a "not too sparse" order sequence, preferably of constant sparseness, and n_1 be small, so that $2n-1 \le n_1 n_2 = O(n)$.

We close this subsection with introducing some technical notation. For a function $f : \mathbb{N} \to \mathbb{N}$ such that $\limsup_{n\to\infty} f(n) = \infty$, we will denote by $f^{\vee}(n)$ the minimal value $f(i)$ over all integer solutions i of the inequality $i \cdot f(i) \ge n$. For example, $n^{\vee} = \lceil \sqrt{n} \rceil$, $\left(\frac{n}{\log n}\right)^{\vee} \sim \sqrt{\frac{2n}{\log n}}$ for $n \ge 2$,[2] and for $q \ge 2$, $(\log_q n)^{\vee} \sim \log_q n$ if $n \ge q$.

We will need to restrict the possible values for i in the inequality to be taken from some order sequence.

For a monotonically growing function $f : \mathbb{N} \to \mathbb{N}$ such that $\lim_{n\to\infty} \frac{f(n)}{n} < 1$, we will define $f^{(0)}(n) = n$, and for $i \ge 1$, $f^{(i)}(n) = f^{(i-1)}(f(n))$. For each $n \ge 1$, there exists the value $i = i(n)$ such that $f^{(i-1)}(n) \ne f^{(i)}(n) = f^{(i+1)}(n) = \cdots$. This value will be denoted by $f^*(n)$. For example, $\left(\lceil \frac{n}{2} \rceil\right)^* = \lceil \log n \rceil$, $(\lceil \sqrt{n} \rceil)^* = \lceil \log \log n \rceil$, $(\lceil \log n \rceil)^* = \lceil \log^* n \rceil$.

4.2 Generalized Algorithm for Polynomial Multiplication

The DFT-based algorithm $\mathcal{A}$, the Schönhage-Strassen's and Schönhage's algorithms $\mathcal{B}$, and the Cantor-Kaltofen's algorithm $\mathcal{C}$ are all based on the idea of a field extension with roots of unity of large smooth orders to reduce the polynomial multiplication to many polynomial multiplications of smaller degrees by means of DFT. The natural metaflow of all these algorithms can be generalized as follows in what we call the algorithm $\mathcal{D}$: let $\mathcal{N}$ be an order sequence of constant sparseness over a field k. For two polynomials $a(x)$ and $b(x)$ of degree $n-1$ over k:

Algorithm $\mathcal{D}$

Embed. Choose a polynomial $P_N(x) \in k[x]$ of degree $N = N'N'' \in \mathcal{N}$ with $2n-1 \le N = O(n)$, and switch to multiplication in $A_N := k[x]/P_N(x)$. From this moment consider $a(x)$ and $b(x)$ as elements of A_N. There should be an injective homomorphism $\psi : A_N \to (A_{N'})^{2N''}$ efficiently computable by means of DFTs, where $A_{N'} \cong k[y]/P_{N'}(y)$ for some $P_{N'}(y) \in k[y]$, and $A_{N'}$ contains a principal N''-th (or $2N''$-th) root of unity.
Transform. Compute the DFTs over $A_{N'}$: $\tilde{a} := \psi(a(x))$ and $\tilde{b} := \psi(b(x))$.
Multiply. Compute $2N''$ products $\tilde{c} := \tilde{a} \cdot \tilde{b}$ in $A_{N'}$.
Back-Transform. By means of DFT compute $c(x) = \psi^{-1}(\tilde{c})$, which is the ordinary product of the input polynomials.
Unembed. Reduce the product modulo $P_N(x)$ to return the product in A_N.

Theorem 1. *The algorithm $\mathcal{A}$, the Schönhage-Strassen and Schönhage algorithms $\mathcal{B}$ and Cantor-Kaltofen algorithm $\mathcal{C}$ are instances of the algorithm $\mathcal{D}$.*

[2] By $f(n) \sim g(n)$ we denote $f(n) = (1 \pm o(1))g(n)$.

Proof. To multiply two degree n polynomials over the ground field k with an N-th primitive root of unity for $N = 2^{\lceil \log(2n-1) \rceil}$, $N = O(n)$, set $P_N(x) = x^N - 1$, $N' = 1$, $N'' = N$ and $A_{N'} = k$. Then ψ is the DFT of order $2N$ (which is trivially reduced to N in this case) over k and the algorithm $\mathcal{D}$ turns into the algorithm $\mathcal{A}$.

For a field k of characteristic different from 2, for $\nu = \lceil \log(2n-1) \rceil$ and $N = 2^\nu$, set $P_N(x) = x^N + 1$, $N' = 2^{\lceil \frac{\nu}{2} \rceil}$, and $N'' = 2^{\lfloor \frac{\nu}{2} \rfloor}$. Then ψ is the DFT of order $2N''$ over $A_{N'}$ and the algorithm $\mathcal{D}$ turns into the Schönhage-Strassen's algorithm $\mathcal{B}$ [16].

For $\operatorname{char} k = 2$, set $\nu = \left\lceil \log_3(n - \frac{1}{2}) \right\rceil$, $N = 3^\nu$, and $P_{2N}(x) = x^{2N} + x^N + 1$, $N' = 3^{\lceil \frac{\nu}{2} \rceil}$, and $N'' = 3^{\lfloor \frac{\nu}{2} \rfloor}$. Then ψ is the DFT of order $3N''$ over $A_{N'}$. However, to fetch the entries of the product in $A_{N'}$ by means ψ^{-1}, $2N''$ products of polynomials in $A_{N'}$ are sufficient [15]. Therefore, the algorithm $\mathcal{D}$ turns into the Schönhage's algorithm $\mathcal{B}$.

For an arbitrary field k fix a positive integer $s \neq \operatorname{char} p$ and find the least ν such that $N = \phi(s^\nu) = s^{\nu-1}\phi(s) \geq 2n - 1$, and let $\hat{N} = s^\nu$. Set $P_{\hat{N}}(x) = \Phi_{\hat{N}}(x)$, $N' = \phi(s^{\lfloor \frac{\nu}{2} \rfloor + 1})$, and $N'' = s^{\lceil \frac{\nu}{2} \rceil - 1}$. Then $\psi = \alpha \circ \beta$ where α stands for 2 DFTs of order N'' over A', and β is a linear map $A_{N'}[x] \to A_{N'}[x] \times A_{N'}[x]$ such that $\beta(a(x)) = (a(x), a(\gamma x))$, where γ is an sN''-th principal root of unity in $A_{N'}$, *i.e.*, for $A_{N'} \cong k[y]/\Phi_{\hat{N}'}(y)$, either $\gamma = y$ or $\gamma = y^2$. One can easily show that β and β^{-1} are computable in linear time [5, Section 2]. Therefore, the algorithm $\mathcal{D}$ turns into the Cantor-Kaltofen's algorithm $\mathcal{C}$. □

4.3 Complexity Analysis

From the description of $\mathcal{D}$ we have $L_{\mathcal{D}}(n) = L'_{\mathcal{D}}(N) = 2N''L'_{\mathcal{D}}(N') + 2T(\psi(N)) + T(\psi^{-1}(N))$, where $L'_{\mathcal{D}}(N)$ stands for the complexity of $\mathcal{D}$ computing the product in A_N, $T(\psi(N))$ and $T(\psi^{-1}(N))$ stand for the total complexities of the transformations ψ and ψ^{-1} on inputs of length N respectively.

Theorem 2. *Let the algorithm $\mathcal{D}$ compute the product of two polynomials in A_N in ℓ recursive steps and let $N' = N'_\lambda$ and $N'' = N''_\lambda$ be chosen on the step $\lambda = 1, \ldots, \ell$ ($N'_0 = N$, $N'_\ell = O(1)$), and for $M(N'_\lambda) = \max\{1, \frac{M^*(N'_\lambda)}{N'_\lambda}\}$, where $M^*(N'_\lambda)$ stands for the complexity of multiplication of an element in $A_{N'_\lambda}$ by powers of an N''_λ-th root of unity (which exists in $A_{N'_\lambda}$ by assumption). Then*

$$L'_{\mathcal{D}}(N) = \Theta\left(N \cdot 2^\ell + N \sum_{\lambda=1}^{\ell} 2^{\lambda-1} \cdot M(N'_\lambda) \log N''_\lambda \right), \tag{11}$$

$$L'_{\mathcal{D}}(N) = \Omega\left(N \cdot 2^{(f_k^\vee)^*(N)} + N \sum_{\lambda=1}^{(f_k^\vee)^*(N)-1} 2^{\lambda-1} \log (f_k^\vee)^{(\lambda)}(N) \right). \tag{12}$$

Proof. Consider the total cost of the algorithm with respect to the computational cost of the first step:

$$L'_{\mathcal{D}}(N) = 2N'' \cdot L'_{\mathcal{D}}(N') + \Theta\left(N'' \log N'' \cdot (N' + M^*(N'))\right). \tag{13}$$

This follows from the fact that we need to perform a constant number of DFTs of order N'' over $A_{N'}$. Since N'' is suitable, each DFT requires $\Theta(N'' \log N'')$ additions of elements in $A_{N'}$ and the same number of multiplications by powers of an N''-th principal root of unity. Since $\dim_k A_{N'} = N'$, one addition in $A_{N'}$ takes N' additions in k, and by definition, $M^*(N')$ is the number of operations in k, needed to compute the necessary products by powers of a principal root of unity. Unrolling (13) (by using (13) recursively ℓ times), (11) follows.

To obtain (12) from (11) we use the trivial lower bound $M(N') \geq 1$. We then notice that $N' \geq f_k^\vee(N'')$, therefore, we come to the equality $N''_\ell = O(1)$ not earlier than for $\ell = (f_k^\vee)^*(N)$, by definition of these operations and the lower bound (12) follows. □

Corollary 3. *1. For any fast field k, we have $L_{\mathcal{D}_k}(n) = O(n \log n)$.*

2. For an $o(\log\log n)$-fast field k, we have $L_{\mathcal{D}_k}(n) = o(n \log n \log\log n)$.

3. For an $\Omega(n^{1-o(1)})$-slow field k, we have $L_{\mathcal{D}_k}(n) = \Omega(n \log n \log\log n)$. In particular, $L_{\mathcal{D}_\mathbb{Q}}(n) = \Omega(n \log n \log\log n)$.

Proof. 1. By definition of a fast field, it suffices to take a constant number of steps (in fact, even one step) to extend k with a principal root of unity of a suitable order. This means, that in (11), $\ell = 1$ and $N' = O(1)$. Therefore, $M(N') = O(1)$ and trivially $\log N'' \leq \log N$.

2. In the first step we have $N' = o(\log\log N)$. We always can bound $M(N'_\lambda)$ with N'_λ in (11), and we have $\ell = o(\log^*(\log^* n))$. Bounding the first summand in the sum in (11) by $N \cdot N' \cdot \log N = o(n \log n \log\log n)$, and each next summand by $o(n \cdot 2^{\log^*(\log^* n)} \cdot \log\log n \cdot \log(\log\log n))$, we obtain the statement.

3. For $f_k(n) = \Omega(n^{1-o(1)})$ we have $f_k^\vee(n) = \Omega(n^{\frac{1}{2}-o(1)})$, which by-turn implies $(f_k^\vee)^*(n) = \Omega(\log\log n)$. Each summand in (12) is therefore $\Omega(\log n)$ and the lower bound for $L_{\mathcal{D}_k}(n)$ follows.

We have $f_\mathbb{Q}(n) = \Omega(\frac{n}{\log\log n}) = \Omega(n^{1-o(1)})$ [1, Theorem 8.8.7], therefore $\mathbb{Q}$ is an $\Omega(n^{1-o(1)})$-slow field which implies the lower bound for $L_{\mathcal{D}_\mathbb{Q}}(n)$. □

Example. Let $p_1 = 2$, $p_2 = 3$, $p_3 = 5$, … be the sequence of all prime numbers. We define the field $k \coloneqq \mathbb{Q}(\omega_1, \omega_2, \dots)$, where $\omega_i \in \mathbb{C}$ is a primitive $p_i^{n''_i}$-th root of unity for $n''_i = \lceil 2^{p_i^3}/(p_i \log p_i) \rceil$. We define the order sequence $\{n_i = n'_i n''_i\}_{i\geq 1}$ where $n'_i = p_i$. This order sequence is $O(\sqrt[3]{\log\log n})$-suitable and the complexity of DFT of order n''_i is $O(p_i n''_i \log n''_i)$. To multiply two degree n polynomials over k we pick in the first step of the algorithm $\mathcal{D}$ the least $n_i \geq n$, which implies $n'_i \sim \sqrt[3]{\log\log n} = o(\log\log n)$. Therefore, $\mathcal{D}$ multiplies two degree n polynomials over k in $O(n \log n (\log\log n)^{2/3}) = o(n \log n \log\log n)$ operations.

Note that Theorem 2 does not give any pessimistic lower bound in case of finite fields. Actually, it can give a good upper bound if one can prove existence of order sequences of constant sparseness over finite fields.

Corollary 4. *1. For the prime finite field $\mathbb{F}_p$, $L_{\mathcal{D}_{\mathbb{F}_p}}(n) = \Omega(n \log n)$.*

2. *Assume that there exists an order sequence $\mathcal{N} = \{n_i(p^{n_i} - 1)\}_{i \geq 1}$ of constant sparseness over $\mathbb{F}_p$ and assume that the multiplication by powers of a principal $(p^{n_i} - 1)$-th root of unity in $\mathbb{F}_{p^{n_i}}$ can be performed in $O(n_i)$ time. Then $L_{\mathcal{D}_{\mathbb{F}_p}}(n) = O(n \log n \log^* n)$.*

Proof. 1. We have $f_{\mathbb{F}_p}(n) \sim \log_p n$ since the multiplicative group $\mathbb{F}_p^*$ is cyclic and in the extension field $\mathbb{F}_{p^n}$ of degree n there exists a primitive root of unity of order $p^n - 1$. This means that $f^{\vee}_{\mathbb{F}_p}(n) \sim \log_p n$ and $(f^{\vee}_{\mathbb{F}_p})^*(n) \sim \log^*_p n$, and the statement follows from taking the first summand in (12) which is $\Theta(n \log n)$.

2. From (13) we get

$$L'_{\mathcal{D}_{\mathbb{F}_p}}(N) \leq \frac{2N}{\log_p N} L'_{\mathcal{D}_{\mathbb{F}_p}}(\lceil \log_p N \rceil) + O(N \log N)\,,$$

and the statement follows from the solution of this inequality. □

There are two challenges to find a faster polynomial multiplication algorithm over finite fields. The first challenge is the already mentioned existence of order sequences $\{n_i = n'_i n''_i\}_{i \geq 1}$ of constant sparseness over these fields, where $n''_i = \omega(\text{poly}(n'_i))$ for $i \geq 1$. This conjecture due to Bläser [2] is supported by performance results of the benchmarks of the algorithm from Corollary 2 (to be published separately).

In Remark 2 we showed that, indeed, there exist suitable order sequences, however, they are too sparse for our purposes. The second challenge is the complexity of multiplication by powers of a primitive root of unity in extension fields. However, there are ways to overcome this with slight complexity increase. We recently obtained some progress in this area, and we think that a general improvement for fields of characteristic different from 2 and 0 is possible.

5 Conclusion

We generalized the notion of a DFT-based algorithm for polynomial multiplication, which describes uniformly all currently known fastest algorithms for polynomial multiplication over *arbitrary fields*. We parameterized fields by introducing the notion of the degree function and order sequences and showed upper and lower bounds for DFT-based algorithm in terms of these paremeters.

There is still an important open question whether one can improve the general Schönhage-Strassen's upper bound. As an outcome of this paper we support the general experience that this question is not very easy. In particular, using only known DFT-based techniques will unlikely help much in case of arbitrary fields, in particular for the case of the rational field, as they did for the complexity of integer multiplication.

Acknowledgements

I would like to thank Markus Bläser for the problem setting and a lot of motivating discussions and anonymous referees for many important improvement suggestions.

References

1. Bach, E., Shallit, J.: Algorithmic Number Theory, Cambridge, MA, vol. 1 (1996)
2. Bläser, M.: Private communication (2010)
3. Bürgisser, P., Clausen, M., Shokrollahi, A.: Algebraic Complexity Theory. Springer, Berlin (1997)
4. Bürgisser, P., Lotz, M.: Lower bounds on the bounded coefficient complexity of bilinear maps. J. ACM 51(3), 464–482 (2004)
5. Cantor, D.G., Kaltofen, E.: On fast multiplication of polynomials over arbitrary algebras. Acta Informatica 28, 693–701 (1991)
6. Chudnovsky, D., Chudnovsky, G.: Algebraic complexities and algebraic curves over finite fields. Journal of Complexity 4, 285–316 (1988)
7. Clausen, M., Baum, U.: Fast Fourier Transforms. Wissenschaftsverlag Mannheim-Leipzig-Wien-Zürich (1993)
8. Cooley, J.W., Tukey, J.W.: An algorithm for the machine calculation of complex Fourier series. Math. Comput. 19, 297–301 (1965)
9. De, A., Kurur, P.P., Saha, C., Saptharishi, R.: Fast integer multiplication using modular arithmetic. In: Proceedings of the 40th ACM STOC 2008 Conference, pp. 499–506 (2008)
10. Fürer, M.: Faster Integer Multiplication. In: Proceedings of the 39th ACM STOC 2007 Conference, pp. 57–66 (2007)
11. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra, 2nd edn. Cambridge University Press, New York (2003)
12. Kaminski, M.: An algorithm for polynomial multiplication that does not depend on the ring of constants. J. Algorithms 9, 137–147 (1988)
13. Pan, V.Y.: Simple Multivariate Polynomial Multiplication. J. Symbolic Computation 18, 183–186 (1994)
14. Rader, C.M.: Discrete Fourier transforms when the number of data samples is prime. Proc. IEEE 56, 1107–1108 (1968)
15. Schönhage, A.: Schnelle Multiplikation von Polynomen über Körpern der Charakteristic 2. Acta Informatica 7, 395–398 (1977)
16. Schönhage, A., Strassen, V.: Schnelle Multiplikation großer Zahlen. Computing 7, 281–292 (1971)
17. Shparlinski, I.E., Tsfasman, M.A., Vladut, S.G.: Curves with many points and multiplication in finite fields. In: Lecture Notes in Math., vol. 1518, pp. 145–169. Springer, Berlin (1992)
18. Strassen, V.: Vermeidung von Divisionen. Crelles J. Reine Angew. Math. 264, 184–202 (1973)

Kolmogorov Complexity as a Language[*]

Alexander Shen[**]

LIF Marseille, CNRS & Univ. Aix–Marseille
alexander.shen@lif.univ-mrs.fr

Abstract. The notion of Kolmogorov complexity (=the minimal length of a program that generates some object) is often useful as a kind of language that allows us to reformulate some notions and therefore provide new intuition. In this survey we provide (with minimal comments) many different examples where notions and statements that involve Kolmogorov complexity are compared with their counterparts not involving complexity.

1 Introduction

The notion of Kolmogorov complexity is often used as a tool; one may ask, however, whether it is indeed a powerful technique or just a way to present the argument in a more intuitive way (for people accustomed to this notion).

The goal of this paper is to provide a series of examples that support both viewpoints. Each example shows some statements or notions that use complexity, and their counterparts that do not mention complexity. In some cases these two parts are direct translations of each other (and sometimes the equivalence can be proved), in other cases they just have the same underlying intuition but reflect it in different ways.

Hoping that most readers already know what is Kolmogorov (algorithmic, description) complexity, we still provide a short reminder to fix notation and terminology. The complexity of a bit string x is the minimal length of a program that produces x. (The programs are also bit strings; they have no input and may produce binary string as output.) If $D(p)$ is the output of program p, the complexity of string x with respect to D is defined as $K_D(x) = \inf\{|p|\colon D(p) = x\}$. This definition depends on the choice of programming language (i.e., its interpreter D), but we can choose an optimal D that makes K_D minimal (up to $O(1)$ constant). Fixing some optimal D, we call $K_D(x)$ the *Kolmogorov complexity* of x and denote it by $K(x)$.

A technical clarification: there are several different versions of Kolmogorov complexity; if we require the programming language to be self-delimiting or

[*] Supported in part by NAFIT ANR-08-EMER-008-01 grant. Author is grateful to all the participants of Kolmogorov seminar at Moscow State University and to his LIF/ESCAPE colleagues. Many of the results covered in this survey were obtained (or at least inspired) by Andrej Muchnik (1958–2007).

[**] On leave from IITP, RAS, Moscow.

prefix-free (no program is a prefix of another one), we got *prefix* complexity usually denoted by $K(x)$; without this requirement we get *plain* complexity usually denoted by $C(x)$; they are quite close to each other (the difference is $O(\log n)$ for n-bit strings and usually can be ignored).

Conditional complexity of a string x given condition y is the minimal length of a program that gets y as input and transforms it into x. Again we need to chose an optimal programming language (for programs with input) among all languages. In this way we get *plain conditional complexity* $C(x|y)$; there exists also a prefix version $K(x|y)$.

The value of $C(x)$ can be interpreted as the "amount of information" in x, measured in bits. The value of $C(x|y)$ measures the amount of information that exists in x but not in y, and the difference $I(y:x) = C(x) - C(x|y)$ measures the amount of information in y about x. The latter quantity is almost commutative (classical Kolmogorov – Levin theorem, one of the first results about Kolmogorov complexity) and can be interpreted as "mutual information" in x and y.

2 Foundations of Probability Theory

2.1 Random Sequences

One of the motivations for the notion of description complexity was to define randomness: n-bit string is random if it does not have regularities that allow us to describe it much shorter, i.e., if its complexity is close to n. For finite strings we do not get a sharp dividing line between random and non-random objects; to get such a line we have to consider infinite sequences. The most popular definition of random infinite sequences was suggested by Per Martin-Löf. In terms of complexity one can rephrase it as follows: bit sequence $\omega_1\omega_2\ldots$ is random if $K(\omega_1\ldots\omega_n) \geq n - c$ for some c and for all n. (This reformulation was suggested by Chaitin; the equivalence was proved by Schnorr and Levin. See more in [14,20].)

Note that in classical probability theory there is no such thing as an individual random object. We say, for example, that randomly generated bit sequence $\omega_1\omega_2\ldots$ satisfies the strong law of large numbers (has limit frequency $\lim(\omega_1 + \ldots + \omega_n)/n$ equal to 1/2) almost surely, but this is just a measure-theoretic statement saying that the set of all ω with limit frequency 1/2 has measure 1. This statement (SLLN) can be proved by using Stirling formula for factorials or Chernoff bound.

Using the notion of Martin-Löf randomness, we can split this statement into two: (1) every Martin-Löf random sequence satisfies SLLN; and (2) the set of Martin-Löf random sequences has measure 1. The second part is a general statement about Martin-Löf randomness (and is easy to prove). The statement (1) can be proved as follows: if the frequency of ones in a long prefix of ω deviates significantly from 1/2, this fact can be used to compress this prefix, e.g., using arithmetic coding or some other technique (Lempel–Ziv compression can be also used), and this is impossible for a random sequence according to the definition.

(In fact this argument is a reformulation of a martingale proof for SLLN.)

Other classical results (e.g., the law of iterated logarithm, ergodic theorem) can be also presented in this way.

2.2 Sampling Random Strings

In the proceeding of this conference S. Aaronson proves a result that can be considered as a connection between two meanings of the word "random" for finite strings. Assume that we bought some device which is marketed as a random number generator. It has some physical source of randomness inside. The advertisement says that, being switched on, this device produces an n-bit random string. What could be the exact meaning of this sentence?

There are two ways to understand it. First: the output distribution of this machine is close to the uniform distribution on n-bit strings. Second: with high probability the output string is random (=incompressible). The paper of Aaronson establishes some connections between these two interpretations (using some additional machinery).

3 Counting Arguments and Existence Proofs

3.1 A Simple Example

Kolmogorov complexity is often used to rephrase counting arguments. We give a simple example (more can be found in [14]).

Let us prove by counting that there exists an $n \times n$ bit matrix without $3\log n \times 3\log n$ uniform minors. (We obtain minors by selecting some rows and columns; the minor is *uniform* if all its elements are the same.)

Counting: Let us give an upper bound for the number of matrices with uniform minors. There are at most $n^{3\log n} \times n^{3\log n}$ positions for a minor (we select $3\log n$ rows and $3\log n$ columns). For each position we have 2 possibilities for the minor (zeros or ones) and $2^{n^2-(3\log n)^2}$ possibilities for the rest, so the total number of matrices with uniform minors does not exceed

$$n^{3\log n} \cdot n^{3\log n} \cdot 2 \cdot 2^{n^2-9\log^2 n} = 2^{n^2-3\log^2 n+1} < 2^{n^2},$$

so there are matrices without uniform minors.

Kolmogorov complexity: Let us prove that incompressible matrix does not have uniform minors. In other words, let us show that matrix with a uniform minor is compressible. Indeed, while listing the elements of such a matrix we do not need to specify all $9\log^2 n$ bits in the uniform minor individually. Instead, it is enough to specify the numbers of the rows of the minor ($3\log n$ numbers; each contains $\log n$ bits) as well as the numbers of columns (this gives together $6\log^2 n$ bits), and to specify the type of the minor (1 bit), so we need only $6\log^2 n + 1 \ll 9\log^2 n$ bits (plus the bits outside the minors, of course).

3.2 One-Tape Turing Machines

One of the first results of computational complexity theory was the proof that some simple operations (checking symmetry or copying) require quadratic time when performed by one-tape Turing machine. This proof becomes very natural if presented in terms of Kolmogorov complexity.

Assume that initially some string x of length n is written on the tape (followed by the end-marker and empty cells). The task is to copy x just after the marker (Fig. 1).

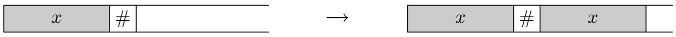


Fig. 1. Copying a bit string x

It is convenient to consider a special case of this task when the first half of x is empty (Fig. 2) and the second half y is an incompressible string of length $n/2$.

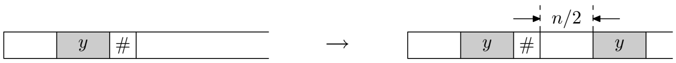


Fig. 2. Special case: the first half of x is empty

To copy y, our machine has to move $n/2$ bits of information across the gap of length $n/2$. Since the amount of information carried by the head of TM is fixed ($\log m$ bits for TM with m states), this requires $\Omega(n^2)$ steps (the hidden constant depends on the number of states).

The last statement can be formalized as follows. Fix some borderline inside the gap and install a "customs office" that writes down the states of TM when it crosses this border from left to right. This record (together with the office position) is enough to reconstruct y (since the behavior of TM on the right of the border is determined by this record). So the record should be of $\Omega(n)$ size. This is true for each of $\Omega(n)$ possible positions of the border, and the sum of the record lengths is a lower bound for the number of steps.

3.3 Forbidden Patterns and Everywhere Complex Sequences

By definition the prefixes of a random sequence have complexity at least $n - O(1)$ where n is the length. Can it be true for all substrings, not only prefixes? No: if it is the case, the sequence at least should be random, and random sequence contains every combination of bits as a substring.

However, Levin noted that the weaker condition $C(x) > \alpha|x| - O(1)$ can be satisfied for all substrings (for any fixed $\alpha < 1$). Such a sequence can be called *α-everywhere complex* sequence. Levin suggested a proof of their existence using some properties of Kolmogorov complexity [6].

The combinatorial counterpart of Levin's lemma is the following statement: let $\alpha < 1$ be a real number and let F be a set of strings that contains at most

$2^{\alpha n}$ strings of length n. Then there exists a constant c and a sequence ω that does not have substrings of length greater than c that belong to F.

It can be shown that this combinatorial statement is equivalent to the original formulation (so it can be formally proved used Kolmogorov complexity); however, there are other proofs, and the most natural one uses Lovasz local lemma. (See [19].)

3.4 Gilbert–Varshamov Bound and Its Generalization

The main problem of coding theory is to find a code with maximal cardinality and given distance. This means that for a given n and given d we want to find some set of n-bit strings whose pairwise Hamming distances are at least d. The strings are called code words, and we want to have as many of them as possible. There is a lower bound that guarantees the existence of large code, called Gilbert–Varshamov bound.

The condition for Hamming distances guarantees that few (less than $d/2$) bit errors during the transmission do not prevent us from reconstructing the original code word. This is true only for errors that change some bits; if, say, some bit is deleted and some other bit is inserted in a different place, this kind of error may be irreparable.

It turns out that we can replace Hamming distance by information distance and get almost the same bound for the number of codewords. Consider some family of n-bit strings $\{x_1, x_2, \ldots\}$. We say that this family is *d-separated*, if $C(x_i|x_j) \geq d$ for $i \neq j$. This means that simple operations of any kind (not only bit changes) cannot transform x_j to x_i. Let us show that for every d there exists a d-separated family of size $\Omega(2^{n-d})$. Indeed, let us choose randomly strings $x_1, \ldots, x_N$ of length n. (The value of N will be chosen later.) For given i and j the probability of the event $C(x_i|x_j) < d$ is less than $2^d/2^n$. For given i the probability that x_i is not separated from *some* x_j (in any direction) does not exceed $2N \cdot 2^d/2^n$, so the expected number of x_i that are "bad" in this sense is less than $2N^2 \cdot 2^d/2^n$. Taking $N = \Omega(2^{n-d})$, we can make this expectation less than $N/2$. Then we can take the values of $x_1, \ldots, x_N$ that give less that $N/2$ bad x_i and delete all the bad x_i, thus decreasing N at most twice. The decreased N is still $\mathit{\Omega}(2^{n-d})$.

It is easy to see that the Gilbert–Varshamov bound (up to some constant) is a corollary of this simple argument. (See [22] for more applications of this argument.)

4 Complexity and Combinatorial Statements

4.1 Inequalities for Kolmogorov Complexity and Their Combinatorial Meaning

We have already mentioned Kolmogorov–Levin theorem about the symmetry of algorithmic information. In fact, they proved this symmetry as a corollary of the following result: $C(x, y) = C(x) + C(y|x) + O(\log n)$. Here x and y are strings of length at most n and $C(x, y)$ is the complexity of some computable encoding of the pair (x, y).

The simple direction of this inequality, $C(x, y) \le C(x) + C(y|x) + O(\log n)$, has equally simple combinatorial meaning. Let A be a finite set of pairs (x, y). Consider the first projection of A, i.e., the set $A_X = \{x: \exists y\,(x, y) \in A\}$. For each x in A_X we also consider the xth section of A, i.e., the set $A_x = \{y: (x, y) \in A\}$. Now the combinatorial counterpart for the inequality can be formulated as follows: if $\#A_X \le 2^k$ and $\#A_x \le 2^l$ for every x, then $\#A \le 2^{k+l}$. (To make the correspondence more clear, we can reformulate the inequality as follows: if $C(x) \le k$ and $C(y|x) \le l$, then $C(x, y) \le k + l + O(\log n)$.)

The more difficult direction, $C(x, y) \ge C(x) + C(y|x) - O(\log n)$, also has a combinatorial counterpart, though more complicated. Let us rewrite this inequality as follows: for every integers k and l, if $C(x, y) \le k + l$, then either $C(x) \le k + O(\log n)$ or $C(y|x) \le l + O(\log n)$. It is easy to see that this statement is equivalent to the original one. Now we can easily guess the combinatorial counterpart: if A is a set of pairs that has at most 2^{k+l} elements, then one can cover it by two sets A' and A'' such that $\#A'_X \le 2^k$ and $\#A''_x \le 2^l$ for every x.

Kolmogorov–Levin theorem implies also the inequality $2C(x, y, z) \le C(x, y) + C(y, z) + C(x, z)$. (Here are below we omit $O(\log n)$ terms, where n is an upper bound of the length for all strings involved.) Indeed, $C(x, y, z) = C(x, y) + C(z|x, y) = C(y, z) + C(x|y, z)$. So the inequality can be rewritten as $C(z|x, y) + C(x|y, z) \le C(x, z)$. It remains to note that $C(x, z) = C(x) + C(z|x)$, that $C(z|x, y) \le C(z|x)$ (more information in the condition implies smaller complexity), and that $C(x|y, z) \le C(x)$ (condition can only help).

The combinatorial counterpart (and the consequence of the inequality about complexities) says that for $A \subset X \times Y \times Z$ we have $(\#A)^2 \le \#A_{X,Y} \cdot \#A_{X,Z} \cdot \#A_{Y,Z}$, where $A_{X,Y}$ is the projection of A onto $X \times Y$, i.e., the set of all pairs (x, y) such that $(x, y, z) \in A$ for some $z \in Z$, etc. In geometric terms: if A is a 3-dimensional body, then the square of its volume does not exceed the product of areas of three its projections (onto three orthogonal planes).

4.2 Common Information and Graph Minors

We have defined the mutual information in two strings a, b as $I(a : b) = C(b) - C(b|a)$; it is equal (with logarithmic precision) to $C(a) + C(b) - C(a, b)$. The easiest way to construct some strings a and b that have significant amount of mutual information is to take overlapping substrings of a random (incompressible) string; it is easy to see that the mutual information is close to the length (and complexity) of their overlap.

We see that in this case the mutual information is not an abstract quantity, but is materialized as a string (the common part of a and b). The natural question arises: is it always the case? i.e., is it possible to find for every pair a, b some string x such that $C(x|a) \approx 0$, $C(x|b) \approx 0$ and $C(x) \approx I(a : b)$?

It turns out that it is not always the case (as found by Andrej Muchnik [5] in Kolmogorov complexity setting and earlier by Gács and Körner [7] in Shannon information setting which we do not describe here — it is not that simple).

The combinatorial counterpart of this question: consider a bipartite graph with (approximately) 2^α vertices on the left and 2^β vertices on the right; assume also that this graph is almost uniform (all vertices in each part have approximately the same degree). Let 2^γ be the total number of edges. A typical edge connects some vertex a on the left and some vertex b on the right, and corresponds to a pair of complexity γ whose first component a has complexity α and second component b has complexity β, so the "mutual information" in this edge is $\delta = \alpha + \beta - \gamma$. The question whether this information can be extracted corresponds to the following combinatorial question: can all (or most) edges of the graph be covered by (approximately) 2^δ minors of size $2^{\alpha-\delta} \times 2^{\beta-\delta}$? (Such a minor connects some $2^{\alpha-\delta}$ vertices on the left with $2^{\beta-\delta}$ vertices on the right.)

For example, consider some finite field F of size 2^n and a plane over this field (i.e., two-dimensional vector space). Consider a bipartite graph whose left vertices are points on this plane, right vertices are lines, and edges correspond to incident pairs. We have about 2^{2n} vertices is each part, and about 2^{3n} edges. This graph does not have 2×2 minors (two different points on a line determine it uniquely). Using this property, one can show that $M \times M$ minor could cover only $O(M\sqrt{M})$ edges. (Assume that M vertices on the left side of such a minor have degrees $d_1, \ldots, d_M$ in the minor. Then for ith vertex on the left there are $\Omega(d_i^2)$ pairs of neighbor vertices on the right, and all these pairs are different, so $\sum d_i^2 \leq O(M^2)$; Cauchy inequality then implies that $\sum d_i \leq O(M\sqrt{M})$, and this sum is the number of edges in the minor).

Translating this argument in the complexity language, we get the following statement: for a random pair (a, b) of incident point and line, the complexity of a and b is about $2n$, the complexity of the pair is about $3n$, the mutual information is about n, but it is not extractable: there is no string x of complexity n such that $C(x|a)$ and $C(x|b)$ are close to zero. In fact, one can prove that for such a pair (a, b) we have $C(x) \leq 2C(x|a) + 2C(x|b) + O(\log n)$ for all x.

4.3 Almost Uniform Sets

Here is an example of Kolmogorov complexity argument that is difficult to translate to combinatorial language (though one may find a combinatorial proof based on different ideas). Consider the set A of pairs. Let us compare the maximal size of its sections A_x and the average size (that is equal to $\#A/\#A_X$; we use the same notation as in section 4.1); the maximal/average ratio will be called X-nonuniformity of A. We can define Y-nonuniformity in the same way.

Claim: *every set A of pairs having cardinality N can be represented as a union of* $\operatorname{polylog}(N)$ *sets whose X- and Y-nonuniformity is bounded by* $\operatorname{polylog}(N)$.

Idea of the proof: consider for each pair $(x, y) \in A$ a quintuple of integers

$$p(x, y) = \langle C(x), C(y), C(x|y), C(y|x), C(x, y)\rangle$$

where all complexities are taken with additional condition A. Each element (x_0, y_0) in A is covered by the set $U(x_0, y_0)$ that consists of all pairs (x, y) for which $p(x, y) \le p(x_0, y_0)$ (coordinate-wise). The number of elements in $U(x_0, y_0)$ is equal to $2^{C(x_0,y_0)}$ up to polynomial in N factors. Indeed, it cannot be greater because $C(x, y) \le C(x_0, y_0)$ for all pairs $(x, y) \in U(x_0, y_0)$. On the other hand, the pair (x_0, y_0) can be described by its ordinal number in the enumeration of all elements of $U(x_0, y_0)$. To construct such an enumeration we need to know only the set A and $p(x_0, y_0)$. The set A is given as a condition, and $p(x_0, y_0)$ has complexity $O(\log N)$. So if the size of $U(x_0, y_0)$ were much less than $2^{C(x_0,y_0)}$, we would get a contradiction.

Similar argument shows that projection $U(x_0, y_0)_X$ has about $2^{C(x_0)}$ elements. Therefore, the average section size is about $2^{C(x_0,y_0)-C(x_0)}$; and the maximal section size does not exceed $C(y_0|x_0)$ since $C(y|x) \le C(y_0|x_0)$ for all $(x, y) \in U(x_0, y_0)$. It remains to note that $C(y_0|x_0) \approx C(x_0, y_0) - C(x_0)$ according to Kolmogorov–Levin theorem, and that there are only polynomially many different sets $U(x, y)$.

Similar argument can be applied to sets of triples, quadruples etc. For a combinatorial proof of this result (in a stronger version) see [1].

5 Shannon Information Theory

5.1 Shannon Coding Theorem

A random variable ξ that has k values with probabilities $p_1, \ldots, p_k$, has *Shannon entropy* $H(\xi) = \sum_i p_i(-\log p_i)$. Shannon coding theorem (in its simplest version) says that if we want to transmit a sequence of N independent values of ξ with small error probability, messages of $NH(\xi) + o(N)$ bits are enough, while messages of $NH(\xi) - o(N)$ bits will lead to error probability close to 1.

Kolmogorov complexity reformulation: *with probability close to* 1 *the sequence of* N *independent values of* ξ *has complexity* $NH(\xi) + o(N)$.

5.2 Complexity, Entropy and Group Size

Complexity and entropy are two ways of measuring the amount of information (cf. the title of the Kolmogorov's paper [11] where he introduced the notion of complexity). So it is not surprising that there are many parallel results. There are even some "meta-theorems" that relate both notions. A. Romashchenko [8] has shown that the linear inequalities that relate complexities of $2^n - 1$ tuples made of n strings $a_1, \ldots, a_n$, are the same as for Shannon entropies of tuples made of n random variables.

In fact, this meta-theorem can be extended to provide combinatorial equivalents for complexity inequalities [18]. Moreover, in [4] it is shown that the same class of inequalities appears when we consider cardinalities of subgroups of some finite group and their intersections!

5.3 Muchnik's Theorem

Let a and b be two strings. Imagine that somebody knows b and wants to know a. Then one needs to send at least $C(a|b)$ bits of information, i.e., the shortest program that transforms b to a. However, if we want the message to be not only short, but also simple relative to a, the shortest program may not work. Andrej Muchnik [15] has shown that it is still possible: *for every two strings a and b of length at most n there exists a string x such that $C(x) \leq C(a|b) + O(\log n)$, $C(a|x, b) = O(\log n)$, and $C(x|a) = O(\log n)$*. This result probably is one of the most fundamental discoveries in Kolmogorov complexity theory of the last decade. It corresponds to Wolf–Slepyan theorem in Shannon information theory; the latter says that for two dependent random variables α and β and N independent trials of this pair one can (with high probability) reconstruct $\alpha_1, \ldots, \alpha_N$ from $\beta_1, \ldots, \beta_N$ and some message that is a function of $\alpha_1, \ldots, \alpha_N$ and has bit length close to $NH(\alpha|\beta)$. However, Muchnik and Wolf–Slepyan theorem do not seem to be corollaries of each other (in any direction).

5.4 Romashchenko's Theorem

Let α, β, γ be three random variables. The mutual information in α and β when γ is known is defined as $I(\alpha : \beta|\gamma) = H(\alpha, \gamma) + H(\beta, \gamma) + H(\alpha, \beta, \gamma) - H(\gamma)$. It is equal to zero if and only if α and β are conditionally independent for every fixed value of γ.

One can show the following: *If $I(\alpha : \beta|\gamma) = I(\alpha : \gamma|\beta) = I(\beta : \gamma|\alpha) = 0$, then one can extract all the common information from α, β, γ in the following sense*: *there is a random variable χ such that $H(\chi|\alpha) = H(\chi|\beta) = H(\chi|\gamma) = 0$ and α, β, γ are independent random variables when χ is known.* (The latter statement can be written as $I(\alpha : \beta\gamma|\chi) = I(\beta : \alpha\gamma|\chi) = I(\gamma : \alpha\beta|\chi) = 0$.)

In algebraic terms: if in a 3-dimensional matrix with non-negative elements all its 2-dimensional sections have rank 1, then (after a suitable permutation for each coordinate) it is made of blocks that have tensor rank 1. (Each block corresponds to some value of χ.)

Romashchenko proved [17] a similarly looking result for Kolmogorov complexity: if a, b, c are three strings such that $I(a : b|c)$, $I(b : c|a)$ and $I(a : c|b)$ are close to zero, then there exists x such that $C(x|a)$, $C(x|b)$, $C(x|c)$ are close to zero and strings a, b, c are independent when x is known, i.e., $I(a : bc|x)$, $I(b : ac|x)$ and $I(c : ab|x)$ are close to zero.

This theorem looks like a direct translation of the information theory result above. However, none of these results looks a corollary of the other one, and Romashchenko's proof is a very ingenious and nice argument that has nothing to do with the rather simple proof of the information-theoretic version.

6 Computability (Recursion) Theory

6.1 Simple Sets

Long ago Post defined *simple* set as (recursively) enumerable set whose complement is infinite but does not contain an infinite enumerable set (see, e.g., [16], Sect. 8.1). His example of such a set is constructed as follows: let W_i be the ith enumerable set; wait until a number $j > 2i$ appears in W_i and include first such number j into the enumeration. In this way we enumerate some set S with infinite complement (S may contain at most n integers less than $2n$); on the other hand, S intersects any infinite enumerable set W_i, because W_i (being infinite) contains some numbers greater than $2i$.

It is interesting to note that one can construct a natural example of a simple set using Kolmogorov complexity. Let us say that a string x is simple if $C(x) < |x|/2$. The set S of simple strings is enumerable (a short program can be discovered if it exists). The complement of S (the set of "complex" strings) is infinite since most n-bit strings are incompressible and therefore non-simple. Finally, if there were an infinite enumerable set $x_1, x_2, \ldots$ of non-simple strings, the algorithm "find the first x_i such that $|x_i| > 2n$" will describe some string of complexity at least n using only $\log n + O(1)$ bits (needed for the binary representation of n).

Similar argument, imitating Berry's paradox, was used by Chaitin to provide a proof for Gödel incompleteness theorem (see Sect. 7.2). Note also a (somewhat mystical) coincidence: the word "simple" appears in two completely different meanings, and the set of all simple strings turns out to be simple.

6.2 Lower Semicomputable Random Reals

A real number α is *computable* if there is an algorithm that computes rational approximations to α with any given precision. An old example of E. Specker shows that a computable series of non-negative rationals can have a finite sum that is not computable. (Let $\{n_1, n_2, \ldots\}$ be a computable enumeration without repetitions of an enumerable undecidable set K; then $\sum_i 2^{-n_i}$ is such a series.) Sums of computable series with non-negative rational terms are called *lower semicomputable* reals.

The reason why the limit of a computable series is not computable is that the convergence is not effective. One can ask whether one can somehow classify how ineffective the convergence is. There are several approaches. R. Solovay introduced some reduction on lower semicomputable reals: $\alpha \preceq \beta$ if $\alpha + \gamma = c\beta$ for some lower semicomputable γ and some rational $c > 0$. Informally, this means that α converges "better" than β (up to a constant c). This partial quasi-ordering has maximal elements called *Solovay complete* reals. It turned out (see [3,13]) that Solovay complete reals can be characterized as lower semicomputable reals whose binary expansion is a random sequence.

Another characterization: we may consider the *modulus of convergence*, i.e., a function that for given n gives the first place where the tail of the series becomes less than 2^{-n}. It turns out that computable series has a random sum if and only if the modulus of convergence grows faster than $BP(n - O(1))$ where $BP(k)$ is the maximal computation time for all terminating k-bit self-delimited programs.

7 Other Examples

7.1 Constructive Proof of Lovasz Local Lemma

Lovasz local lemma considers a big (unbounded) number of events that have small probability and are mostly independent. It guarantees that sometimes (with positive probability, may be very small) none of this events happens. We do not give the exact statement but show a typical application: *any CNF made of k-literal clauses where each clause has $t = o(2^k)$ neighbors, is satisfiable.* (Neighbors are clauses that have a common variable.)

The original proof by Lovasz (a simple induction proving some lower bound for probabilities) is not constructive in the sense that it does not provide any algorithm to find the satisfying assignment (better than exhaustive search). However, recently Moser discovered that naive algorithm: "resample clauses that are false until you are done" converges in polynomial time with high probability, and this can be explained using Kolmogorov complexity. Consider the following procedure (Fig. 3; by *resampling* a clause we mean that all variables in this clause get fresh random values). It is easy to see that this procedure satisfies the specification if terminates (induction).

```
{Clause C is false}
procedure Fix (C: clause)=
   resample (C);
   for all neighbor clauses C' of C: if C' is false then Fix(C')
{Clause C is true; all the clauses that were true
       before the call Fix(C), remain true}
```

Fig. 3. Moses' resampling algorithm

The pre- and post-conditions guarantee that we can find a satisfying assignment applying this procedure to all the clauses (assuming the termination). It remains to show that with high probability this procedure terminates in a polynomial time. Imagine that Fix(X) was called for some clause X and this call does not terminate for a long time. We want to get a contradiction. Crucial observation: *at any moment of the computation the sequence of recursive calls made during the execution* (i.e., the ordered list of clauses C for which Fix(C) was called) *together with the current values of all variables determine completely the random bits used for resampling.* (This will allow us to compress the sequence of random bits used for resampling and get a contradiction.) Indeed, we can roll back the computation; note that for every clause in the CNF there is exactly one combination of its variables that makes it false, and our procedure is called only if the clause is false, so we know the values before each resampling.

Now we estimate the number of bits needed to describe the sequence of recursive calls. These calls form a tree. Consider a path that visits all the vertices of this tree (=calls) in the usual way, following the execution process (going from a calling instance to a called one and returning back). Note that called procedure corresponds to one of t neighbors of the calling one, so each step down in the tree can be described by $1 + \log t$ bits (we need to say that it is a step down and specify the neighbor). Each step up needs only 1 bit (since we return to known instance). The number of steps up does not exceed the number of steps down, so we need in total $2 + \log t$ bits per call. Since $t = o(2^k)$ by assumption, we can describe the sequence of calls using $k - O(1)$ bits per call which is less than the number of random bits (k per call), so the sequence of calls cannot be long.

7.2 Berry, Gödel, Chaitin, Raz

Chaitin found (and popularized) a proof of Gödel incompleteness theorem based on the Berry paradox ("the smallest integer not definable by eight words"). He showed that statements of the form "$C(x) > k$" where x is a string and k is a number, can be proved (in some formal theory, e.g., Peano arithmetic) only for bounded values of k. Indeed, if it were not the case, we could try all proofs and for every number n effectively find some string x_n which has guaranteed complexity above n. Informally, x_n is some string provably not definable by n bits. But it can be defined by $\log n + O(1)$ bits ($\log n$ bits are needed to describe n and $O(1)$ bits describe the algorithm transforming n to x_n), so we get a contradiction for large enough n. (The difference with the Berry paradox is that x_n is not the minimal string, just the first one in the proofs enumeration ordering.)

Recently Kritchman and Raz found that another paradox, "Surprise Examination" (you are told that there will be a surprise examination next week: you realize that it cannot be at Saturday, since then you would know this by Friday evening; so the last possible day is Friday, and if it were at Friday, you would know this by Thursday evening, etc.), can be transformed into a proof of second Gödel incompleteness theorem; the role of the day of the examination is played by the number of incompressible strings of length n. (The argument starts as follows: We can prove that such a string exists; if it were only one string, it can be found by waiting until all other strings turn out to be compressible, so we know there are at least two, etc. In fact you need more delicate argument that uses some properties of Peano arithmetic — the same properties as in Gödel's proof.)

7.3 13th Hilbert Problem

Thirteenth Hilbert problem asked whether some specific function (that gives a root of a degree 7 polynomial as a function of its coefficients) can be expressed as a composition of continuous functions of one and two real variables. More than fifty years later Kolmogorov and Arnold showed that the answer to this question is positive: any continuous function of several real arguments can be represented as a composition of continuous functions of one variable and addition. (For other classes instead of continuous function this is not the case.) Recently this question was discussed in the framework of circuit complexity [10].

It has also some natural counterpart in Kolmogorov complexity theory. Imagine that three string a, b, c are written on the blackboard. We are allowed to write any string that is simple (has small conditional complexity) relative to any *two* strings on the board, and can do this several times (but not too many: otherwise we can get any string by changing one bit at a time). Which strings could appear if we follow this rule? The necessary condition: strings that appear are simple relative to (a, b, c). It turns out, however, that it is not enough: some strings are simple relative to (a, b, c) but cannot be obtained in this way. This is not difficult to prove (see [21] for the proof and references); what would be really interesting is to find some specific example, i.e., to give an explicit function with three string arguments such that $f(a, b, c)$ cannot be obtained in the way described starting from random a, b, and c.

7.4 Secret Sharing

Imagine some secret (i.e., password) that should be shared among several people in such a way that some (large enough) groups are able to reconstruct the secret while other groups have no information about it. For example, for a secret s that is an element of the finite field F, we can choose a random element a of the same field and make three shares a, $a + s$ and $a + 2s$ giving them to three participants X, Y, Z respectively. Then each of three participants has no information about the secret s, since each share is a uniformly distributed random variable. On the other hand, any two people together can reconstruct the secret. One can say that this secret sharing scheme implements the access structure $\{\{X, Y\}, \{X, Z\}, \{Y, Z\}\}$ (access structure lists minimal sets of participants that are authorized to know the secret).

Formally, a secret sharing scheme can be defined as a tuple of random variables (one for the secret and one for each participant); the scheme implements some access structure if all groups of participants listed in this structure can uniquely reconstruct the value of the secret, and for all other groups (that do not contain any of the groups listed) their information is independent of the secret. It is easy to see that any access structure can be implemented; the interesting (and open) question is to find how big should be the shares (for a given secret size and a given access structure).

We gave the definition of secret sharing in probability theory framework; however, one can also consider it in Kolmogorov complexity framework. For example, take binary string s as a secret. We may look for three strings x, y, z such that $C(s|x, y)$, $C(s|y, z)$, and $C(s|x, z)$ are very small (compared to the complexity of the secret itself), as well as the values of $I(x : s)$, $I(y : s)$, and $I(z : s)$. The first requirement means that any two participants know (almost) everything about the secret; the second requirement means each participant alone has (almost) no information about it.

The interesting (and not well studied yet) question is whether these two frameworks are equivalent in some sense (the same access structure can be implemented with the same efficiency); one may also ask whether in Kolmogorov setting the possibility of sharing secret s with given access structure and share sizes depends only on the complexity of s. Some partial results were obtained recently by T. Kaced and A. Romashchenko (private communication). The use of Kolmogorov complexity in cryptography is discussed in [2].

7.5 Quasi-cryptography

The notion of Kolmogorov complexity can be used to pose some questions that resemble cryptography (though probably are hardly practical). Imagine that some intelligence agency wants to send a message b to its agent. They know that agent has some information a. So their message f should be enough to reconstruct a from b, i.e., $C(b|a, f)$ should be small. On the other hand, the message f without a should have minimal information about b, so the complexity $C(b|f)$ should be maximal.

It is easy to see that $C(b|f)$ cannot exceed $\min(C(a), C(b))$ because both a and b are sufficient to reconstruct b from f. Andrej Muchnik proved that indeed this bound is tight, i.e., there is some message f that reaches it (with logarithmic precision).

Moreover, let us assume that eavesdropper knows some c. Then we want to make $C(b|c, f)$ maximal. Muchnik showed that in this case the maximal possible value (for f such that $C(b|a, f) \approx 0$) is $\min(C(a|c), C(b|c))$. He also proved a more difficult result that bounds the size of f, at least in the case when a is complex enough. The formal statement of the latter result: *There exists some constant d such that for every strings a, b, c of length at most N such that $C(a|c) \geq C(b|c) + C(b|a) + d \log N$, there exists a string f of length at most $C(b|a) + d \log N$ such that $C(b|a, f) \leq d \log N$ and $C(b|c, f) \geq C(b|c) - d \log N$.*

References

1. Alon, N., Newman, I., Shen, A., Tardos, G., Vereshchagin, N.K.: Partitioning multi-dimensional sets in a small number of “uniform” parts. European Journal of Combinatorics 28(1), 134–144 (2007)
2. Antunes, L., Laplante, S., Pinto, A., Salvador, L.: Cryptographic Security of Individual Instances. In: Desmedt, Y. (ed.) ICITS 2007. LNCS, vol. 4883, pp. 195–210. Springer, Heidelberg (2009)
3. Calude, C.S., Hertling, P.H., Khoussainov, B., Wang, Y.: Recursively Enumerable Reals and Chaitin Ω Numbers. Theoretical Computer Science 255, 125–149 (2001)
4. Chan, T.H., Yeung, R.W.: On a relation between information inequalities and group theory. IEEE Transaction on Information theory 48(7), 1992–1995 (2002)
5. Chernov, A., Muchnik, A.A., Romashchenko, A.E., Shen, A., Vereshchagin, N.K.: Upper semi-lattice of binary strings with the relation “x is simple conditional to y”. Theoretical Computer Science 271(1-2), 69–95 (2002)
6. Durand, B., Levin, L.A., Shen, A.: Complex Tilings. Journal of Symbolic Logic 73(2), 593–613 (2007)

7. Gács, P., Korner, J.: Common Information is Far Less Than Mutual Information. Problems of Control and Information Theory 2(2), 119–162 (1973)
8. Hammer, D., Romashchenko, A.E., Shen, A., Vereshchagin, N.: Inequalities for Shannon Entropy and Kolmogorov Complexity. Journal for Computer and System Sciences 60, 442–464 (2000)
9. Hammer, D., Shen, A.: A Strange Application of Kolmogorov Complexity. Theory of Computing Systems 31(1), 1–4 (1998)
10. Hansen, K.A., Lachish, O., Miltersen, P.B.: Hilbert's Thirteenth Problem and Circuit Complexity. In: Dong, Y., Du, D.-Z., Ibarra, O. (eds.) ISAAC 2009. LNCS, vol. 5878, pp. 153–162. Springer, Heidelberg (2009)
11. Kolmogorov, A.N.: Three approaches to the definition of the concept "quantity of information". Problemy Peredachi Informatsii 1(1), 3–11 (1965) (Russian)
12. Kritchman, S., Raz, R.: The Surprise Examination Paradox and the Second Incompleteness Theorem. Notices of the AMS 75(11), 1454–1458 (2010)
13. Kučera, A., Slaman, T.A.: Randomness and recursive enumerability. SIAM Journal on Computing 31(1), 199–211 (2001)
14. Li, M., Vitanyi, P.: An Introduction to Kolmogorov Complexity and Its Applications, 3rd edn. Springer, Heidelberg (2008)
15. Muchnik, A.: Conditional complexity and codes. Theoretical Computer Science 271(1-2), 97–109 (2002)
16. Rogers Jr., H.: Theory of Recursive Functions and Effective Computability. McGraw-Hill Book Company, New York (1967)
17. Romashchenko, A.E.: A Criterion of Extractability of Mutual Information for a Triple of Strings. Problems of Information Transmission 39(1), 148–157
18. Romashchenko, A.E., Shen, A., Vereshchagin, N.K.: Combinatorial Interpretation of Kolmogorov Complexity. Theoretical Computer Science 271(1-2), 111–123 (2002)
19. Rumyantsev, A.Y., Ushakov, M.A.: Forbidden substrings, kolmogorov complexity and almost periodic sequences. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 396–407. Springer, Heidelberg (2006), arxiv.org/abs/1009.4455
20. Shen, A.: Algorithmic Information theory and Kolmogorov complexity. Uppsala university Technical Report TR2000-034, www.it.uu.se/research/publications/reports/2000-034/2000-034-nc.ps.gz
21. Shen, A.: Decomposition complexity. Journées Automates Cellulaires (Turku), 203–213 (2010), hal-00541921 at archives-ouvertes.fr
22. Ti, Y.-W., Chang, C.-L., Lyuu, Y.-D., Shen, A.: Sets of k-independent strings. International Journal of Foundations of Computer Science 21(3), 321–327 (2010)

Almost k-Wise Independent Sets Establish Hitting Sets for Width-3 1-Branching Programs*

Jiří Šíma and Stanislav Žák

Institute of Computer Science, Academy of Sciences of the Czech Republic,
P.O. Box 5, 18207 Prague 8, Czech Republic
{sima,stan}@cs.cas.cz

Abstract. Recently, an interest in constructing pseudorandom or hitting set generators for restricted branching programs has increased, which is motivated by the fundamental problem of derandomizing space bounded computations. Such constructions have been known only in the case of width 2 and in very restricted cases of bounded width. In our previous work, we have introduced a so-called richness condition which is, in a certain sense, sufficient for a set to be a hitting set for read-once branching programs of width 3. In this paper, we prove that, for a suitable constant C, any almost $C \log n$-wise independent set satisfies this richness condition. Hence, we achieve an explicit polynomial time construction of a hitting set for read-once branching programs of width 3 with the acceptance probability greater than $\sqrt{12/13}$ by using the result due to Alon et al. (1992).

1 Introduction

The relationship between deterministic and probabilistic computations is one of the central issues in complexity theory. This problem can be tackled by constructing polynomial time pseudorandom [10] or *hitting set* generators [6] which, however, belongs to the hardest problems in computer science even for severely restricted computational models. In particular, derandomizing space bounded computations has attracted much interest over a decade. We consider *read-once branching (1-branching) programs* [14] of polynomial size for which pseudorandom generators with seed length $O(\log^2 n)$ have been known for a long time through a result of Nisan [9]. Recently, considerable attention has been paid to improving this to $O(\log n)$ in the constant-width case, which is a fundamental problem with many applications in circuit lower bounds and derandomization [8]. The problem has been resolved for width 2 but the known techniques provably fail for width 3 [2,8], which applies even to hitting set generators [4].

In the case of width 3, we do not know of any significant improvement over Nisan's result except for severely restricted so-called *regular* (oblivious) read-once branching programs of constant width having the in-degree of every vertex equal to 2, for which pseudorandom generators have recently been constructed

* This research was partially supported by projects GA ČR P202/10/1333, MŠMT ČR 1M0545, and AV0Z10300504.

with seed length $O(\log n \log\log n)$ [3,4]. There has also been some recent progress in the case of *permutation* (oblivious) read-once branching programs of bounded width whose edges labeled with 0 (respectively 1) define a one to one mapping for each level-to-level transition [8], for which a pseudorandom generator has been constructed with seed length $O(\log n)$ [7]. In our paper [11], we made the first step for finding a polynomial time constructible hitting set for width 3. Using the result due to Alon et al. [1], we achieved such a construction if an additional, rather technical restriction is imposed on the program structure. For example, this restriction is met if one special pattern of level-to-level transitions in a normalized form of so-called *simple* width-3 1-branching programs is excluded, which covers the regular and permutation cases (see [11] for further details).

In our previous work [12,13], we have introduced a so-called *richness* condition which is independent of the notion of branching programs. In fact, a rich set is a hitting set for special read-once CNFs (or even for the read-once conjunctions of DNFs and CNFs with properly bounded monomial and clause sizes, respectively [12]). Thus, a related line of study concerns pseudorandom generators for read-once formulas, such as read-once DNFs [5]. This richness condition proves to be sufficient in a sense that any rich set extended with all strings within Hamming distance of at most 3 is a hitting set for width-3 1-branching programs with the acceptance probability greater than $\sqrt{12/13}$. In this paper, we prove that, for a suitable constant C, any almost $C\log n$-wise independent set satisfies the richness condition. In the proof, the probability that there is a certain input which ensures the richness of an almost k-wise independent set, is lower bounded by a positive number (e.g. by using the inclusion-exclusion principle). It follows that our result combined with an efficient construction of almost k-wise independent sets, e.g. due to Alon et al. [1], provides a polynomial time construction of a hitting set for width-3 1-branching programs.

The paper is organized as follows. After a brief review of basic definitions regarding branching programs, the richness condition and its sufficiency is presented in Section 2. The main result that any almost $O(\log n)$-wise independent set is rich is formulated in Section 3 where the main steps of the technical proof occupying the subsequent four Sections 4–7 are outlined. Our result is summarized in Section 8.

2 Branching Programs and the Richness Condition

We start with a brief review of basic formal definitions regarding branching programs (see [14] for more information). A *branching program* P on the set of input Boolean variables $X_n = \{x_1, \ldots, x_n\}$ is a directed acyclic multi-graph $G = (V, E)$ that has one *source* $s \in V$ of zero in-degree and, except for *sinks* of zero out-degree, all the *inner* (non-sink) nodes have out-degree 2. In addition, the inner nodes get labels from X_n and the sinks get labels from $\{0, 1\}$. For each inner node, one of the outgoing edges gets the label 0 and the other one gets the label 1. The branching program P computes Boolean function $P : \{0,1\}^n \longrightarrow \{0, 1\}$ as follows. The computational path of P for an input $\mathbf{a} = (a_1, \ldots, a_n) \in$

$\{0,1\}^n$ starts at source s. At any inner node labeled by $x_i \in X_n$, input variable x_i is tested and this path continues with the outgoing edge labeled by a_i to the next node, which is repeated until the path reaches the sink whose label gives the output value $P(\mathbf{a})$. Denote by $P^{-1}(a) = \{\mathbf{a} \in \{0,1\}^n \mid P(\mathbf{a}) = a\}$ the set of inputs for which P outputs $a \in \{0,1\}$. For inputs of arbitrary lengths, infinite families $\{P_n\}$ of branching programs, each P_n for one input length $n \geq 1$, are used. A branching program P is called *read-once* (or shortly *1-branching* program) if every input variable from X_n is tested at most once along each computational path. Here we consider *leveled* branching programs in which each node belongs to a level, and edges lead from level $k \geq 0$ only to the next level $k+1$. We assume that the source of P creates level 0, whereas the last level is composed of all sinks. The maximum number of nodes on one level is called the *width* of P.

Let $\mathcal{P}$ be a class of branching programs and $\varepsilon > 0$ be a real constant. A set of input strings $H \subseteq \{0,1\}^*$ is called an *ε-hitting set* for class $\mathcal{P}$ if for sufficiently large n, for every branching program $P \in \mathcal{P}$ with n input variables,

$$\frac{\left|P^{-1}(1)\right|}{2^n} \geq \varepsilon \quad \text{implies} \quad (\exists\, \mathbf{a} \in H \cap \{0,1\}^n)\, P(\mathbf{a}) = 1\,. \tag{1}$$

Furthermore, we say that a set $A \subseteq \{0,1\}^*$ is *ε-rich* if for sufficiently large n, for any index set $I \subseteq \{1,\ldots,n\}$, for any partition $\{R_1,\ldots,R_r\}$ of I ($r \geq 0$) satisfying

$$\prod_{j=1}^{r} \left(1 - \frac{1}{2^{|R_j|}}\right) \geq \varepsilon\,, \tag{2}$$

and for any $Q \subseteq \{1,\ldots,n\} \setminus I$ such that $|Q| \leq \log n$, for any $\mathbf{c} \in \{0,1\}^n$ there exists $\mathbf{a} \in A \cap \{0,1\}^n$ that meets

$$(\forall\, i \in Q)\, a_i = c_i \text{ and } (\forall\, j \in \{1,\ldots,r\})\, (\exists\, i \in R_j)\, a_i \neq c_i\,. \tag{3}$$

Note that formula (3) can be interpreted as a read-once CNF (each variable occurs at most once) which contains at most logarithmic number of single literals together with clauses whose sizes satisfy (2). Hence, any rich set is a hitting set for such read-once CNFs. In the following theorem, we formulate our previous result [12,13] that the richness condition is, in a certain sense, sufficient for a set to be a hitting set for read-once branching programs of width 3.

Theorem 1 ([12,13]). *If A is $(\delta\,\varepsilon)^{11}$-rich for $\varepsilon > \delta = \sqrt{12/13}$, then $H = \Omega_3(A) = \{\mathbf{a}' \in \{0,1\}^n \mid n \geq 1\,\&\,(\exists \mathbf{a} \in A \cap \{0,1\}^n)\, h(\mathbf{a},\mathbf{a}') \leq 3\}$, where $h(\mathbf{a},\mathbf{a}')$ is the Hamming distance between $\mathbf{a}$ and $\mathbf{a}'$ (i.e. the number bits in which $\mathbf{a}$ and $\mathbf{a}'$ differ), is an ε-hitting set for the class of width-3 read-once branching programs.*

3 Almost k-Wise Independent Sets Are Rich

The following theorem shows that the richness condition introduced in previous Section 2 is satisfied by almost k-wise independent sets. Hence, in order to achieve an explicit polynomial time construction of a hitting set for read-once branching programs of width 3, we can combine Theorem 1 with the result due to Alon et al. [1] who provided simple efficient constructions of almost k-wise independent sets. In particular, for $\beta > 0$ and $k = O(\log n)$ it is possible to construct a (k, β)*-wise independent set* $\mathcal{A} \subseteq \{0,1\}^*$ in time polynomial in $\frac{n}{\beta}$ such that for sufficiently large n and any index set $S \subseteq \{1,\ldots,n\}$ of size $|S| \le k$, the probability that a given $\mathbf{c} \in \{0,1\}^n$ coincides with a string $\mathbf{a} \in \mathcal{A}_n = \mathcal{A} \cap \{0,1\}^n$ on the bit locations from S is almost uniform, that is

$$\left| \frac{|\mathcal{A}_n^S(\mathbf{c})|}{|\mathcal{A}_n|} - \frac{1}{2^{|S|}} \right| \le \beta\,, \tag{4}$$

where $\mathcal{A}_n^S(\mathbf{c}) = \{\mathbf{a} \in \mathcal{A}_n \mid (\forall i \in S)\, a_i = c_i\}$. We will prove that, for suitable k, any almost k-wise independent set is ε-rich.

Theorem 2. *Let $\varepsilon > 0$, C be the least odd integer greater than $(\frac{2}{\varepsilon} \ln \frac{1}{\varepsilon})^2$, and $0 < \beta < \frac{1}{n^{C+3}}$. Then any $(\lceil (C+2)\log n \rceil, \beta)$-wise independent set is ε-rich.*

Proof. Let $\mathcal{A} \subseteq \{0,1\}^*$ be a $(\lceil (C+2)\log n \rceil, \beta)$-wise independent set. We will show that $\mathcal{A}$ is ε-rich. Assume $\{R_1,\ldots,R_r\}$ is a partition of index set $I \subseteq \{1,\ldots,n\}$ satisfying condition (2), and $Q \subseteq \{1,\ldots,n\} \setminus I$ such that $|Q| \le \log n$. In order to show for a given $\mathbf{c} \in \{0,1\}^n$ that there is $\mathbf{a} \in \mathcal{A}_n$ that meets (3) for Q and partition $\{R_1,\ldots,R_r\}$, we will prove that the probability

$$p = \frac{\left| \mathcal{A}_n^Q(\mathbf{c}) \setminus \bigcup_{j=1}^r \mathcal{A}_n^{R_j}(\mathbf{c}) \right|}{|\mathcal{A}_n|} \tag{5}$$

of the event that $\mathbf{a} \in \mathcal{A}_n$ chosen uniformly at random satisfies $\mathbf{a} \in \mathcal{A}_n^Q(\mathbf{c})$ and $\mathbf{a} \notin \mathcal{A}_n^{R_j}(\mathbf{c})$ for every $j = 1,\ldots,r$, is *strictly positive.*

The main idea of the proof lies in lower bounding the probability (5). We briefly comment on the main steps of the proof which are schematically depicted in Figure 1 including references to corresponding sections, lemmas, and equations. In Section 4, we will first modify the partition classes R_j so that their cardinalities are at most logarithmic whereas the classes of small constant cardinalities are merged with Q and also $\mathbf{c}$ is adjusted correspondingly. Lemma 1 then ensures that the probability p from (5) is lower bounded when using these modified classes. Furthermore, Bonferroni inequality (the inclusion-exclusion principle) and the assumption concerning the almost k-wise independence are employed in Section 5 where also the classes of the same cardinality are grouped. In Section 6, we will further reduce the underlying lower bound on p only to a sum over frequent cardinalities of partition classes to which Taylor's theorem is applied in Section 7, whereas a corresponding Lagrange remainder is bounded using the assumption on constant C.

4 Modifications of Partition Classes

We properly modify the underlying partition classes in order to further upper bound their cardinalities by the logarithmic function so that the assumption

Modifications of Partition Classes (Section 4)

- superlogarithmic cardinalities:
  $R'_j \subseteq R_j$ so that $|R'_j| \le \log n$ (6)
- small constant cardinalities:
  $R_\le = \bigcup_{|R'_j|\le\sigma} R'_j$ where σ is a constant (11) & (14)
  $\longrightarrow Q' = Q \cup R_\le$ (16), $c'_i = 1 - c_i$ for $i \in R_\le$ (19)

Lemma 1: $p \ge \frac{\left|\mathcal{A}_n^{Q'}(\mathbf{c}')\setminus\bigcup_{j=1}^{r'}\mathcal{A}_n^{R'_j}(\mathbf{c}')\right|}{|\mathcal{A}_n|}$ (20)

$\big\downarrow$ **Bonferroni inequality**

$$p \ge \sum_{k=0}^{C'}(-1)^k \sum_{1\le j_1<j_2<\cdots<j_k\le r'} \frac{\left|\mathcal{A}_n^{\bigcup_{i=1}^k R'_{j_i}\cup Q'}(\mathbf{c}')\right|}{|\mathcal{A}_n|} \qquad (22)$$

$\big\downarrow$ **almost k-wise independence** (Section 5)

$$p \ge \frac{1}{2^{|Q'|}}\left(\sum_{k=0}^{C'}(-1)^k \sum_{1\le j_1<j_2<\cdots<j_k\le r'} \prod_{i=1}^{k}\frac{1}{2^{\left|R'_{j_i}\right|}} - \frac{\varepsilon'}{8}\right) \qquad (25)\ \&\ (26)$$

Grouping the Same Cardinalities (Lemma 2)

$\sigma < s_1, \ldots, s_{m'} \le \log n$ … cardinalities of R'_j

$r_i = \left|\left\{j\,|, \left|R'_j\right| = s_i\right\}\right|$ … the number of classes of cardinality s_i

$$p > \frac{1}{n^2}\left(\sum_{k=0}^{C'}(-1)^k \sum_{\substack{k_1+\cdots+k_{m'}=k\\ 0\le k_1\le r_1,\ldots,0\le k_{m'}\le r_{m'}}} \prod_{i=1}^{m'}\frac{t_i^{k_i}}{k_i!}\prod_{j=1}^{k_i-1}\left(1-\frac{j}{r_i}\right) - \frac{\varepsilon'}{8}\right) \qquad (30)$$

where $t_i = \frac{r_i}{2^{s_i}}$ (8)

Frequent Cardinalities (Section 6 & Lemma 3)

$r_1 > r_2 > \cdots > r_{m''} > \varrho$ where ϱ is a constant (11) & (12)

$$p > \frac{1}{n^2}\left(\sum_{k=0}^{C'}(-1)^k \sum_{\substack{k_1+\cdots+k_{m''}=k\\ k_1\ge 0,\ldots,k_{m''}\ge 0}} \prod_{i=1}^{m''}\frac{t_i^{k_i}}{k_i!} - \frac{\varepsilon'}{2}\right) \qquad (41)\ \&\ \text{Lemma 4.i}$$

$\big\downarrow$ multinomial theorem

$$p > \tfrac{1}{n^2}\left(\sum_{k=0}^{C'}\frac{\left(-\sum_{i=1}^{m''}t_i\right)^k}{k!} - \frac{\varepsilon'}{2}\right) \qquad (43)$$

$\big\downarrow$ **Taylor's theorem** (Section 7)

$$p > \tfrac{1}{n^2}\left(e^{-\sum_{i=1}^{m''}t_i} - \mathcal{R}_{C'+1}\left(-\sum_{i=1}^{m''}t_i\right) - \tfrac{\varepsilon'}{2}\right) \qquad (44)$$

$\big\downarrow$ (2) $\longrightarrow \sum_{i=1}^{m} t_i < \ln\frac{1}{\varepsilon'}$ (10)
Lagrange remainder $\mathcal{R}_{C'+1}\left(-\sum_{i=1}^{m''}t_i\right) < \frac{\varepsilon'}{4}$ (Lemma 4.ii)

$$p > \tfrac{\varepsilon'}{4n^2} > 0 \qquad (51)$$

Fig. 1. The main steps of the proof

concerning almost $\lceil (C+2)\log n\rceil$-wise independence of $\mathcal{A}$ can be applied in the following Section 5. Thus, we confine ourselves to at most logarithmic-size arbitrary subsets R'_j of partition classes R_j, that is

$$R'_j \begin{cases} = R_j & \text{if } |R_j| \leq \log n \\ \subset R_j \text{ so that } |R'_j| = \lfloor \log n \rfloor & \text{otherwise}, \end{cases} \tag{6}$$

which ensures $R'_j \subseteq R_j$ and $|R'_j| \leq \log n$ for every $j = 1, \ldots, r$. For these new classes, assumption (2) can be rewritten as

$$\prod_{j=1}^{r}\left(1-\frac{1}{2^{|R'_j|}}\right) > \left(1-\frac{1}{2^{\log n}}\right)^{\frac{n}{\log n}} \prod_{|R_j|\leq \log n}\left(1-\frac{1}{2^{|R_j|}}\right)$$
$$> \left(1-\frac{1}{n}\cdot\frac{n}{\log n}\right)\varepsilon = \left(1-\frac{1}{\log n}\right)\varepsilon = \varepsilon', \tag{7}$$

where $\varepsilon' > 0$ is arbitrarily close to ε for sufficiently large n.

Denote by $\{s_1, s_2, \ldots, s_m\} = \{|R'_1|, \ldots, |R'_r|\}$ the set of all cardinalities $1 \leq s_i \leq \log n$ of classes $R'_1, \ldots, R'_r$, and for every $i = 1, \ldots, m$, let $r_i = |\{j \,|\, |R'_j| = s_i\}|$ be the number of classes R'_j having cardinality s_i, that is, $r = \sum_{i=1}^{m} r_i$. Furthermore, we define

$$t_i = \frac{r_i}{2^{s_i}} > 0 \quad \text{for } i = 1, \ldots, m\,. \tag{8}$$

It follows from (7) and (8) that

$$0 < \varepsilon' < \prod_{j=1}^{r}\left(1-\frac{1}{2^{|R'_j|}}\right) = \prod_{i=1}^{m}\left(\left(1-\frac{1}{2^{s_i}}\right)^{2^{s_i}}\right)^{t_i} < e^{-\sum_{i=1}^{m} t_i} \tag{9}$$

implying

$$\sum_{i=1}^{m} t_i < \ln\frac{1}{\varepsilon'}\,. \tag{10}$$

Moreover, we define constants

$$\varrho = \frac{C}{1-\left(1-\frac{\varepsilon'^2}{4(1+\varepsilon'^2)}\right)^{\frac{1}{C}}} > C \geq 1\,, \qquad \sigma = \log\left(\frac{4\varrho\,(1+\varepsilon'^2)}{\varepsilon'^2}\right) \tag{11}$$

which are used for sorting the cardinalities $s_1, \ldots, s_m$ so that

$$r_i > \varrho \text{ and } s_i > \sigma \quad \text{for } \ i = 1, \ldots, m'' \tag{12}$$
$$r_i \leq \varrho \text{ and } s_i > \sigma \quad \text{for } \ i = m''+1, \ldots, m' \tag{13}$$
$$s_i \leq \sigma \quad \text{for } \ i = m'+1, \ldots, m\,. \tag{14}$$

We will further confine ourselves to the first $m' \geq 0$ cardinalities satisfying $s_i > \sigma$ for $i = 1, \ldots, m'$. Without loss of generality, we can also sort the corresponding partition classes so that $|R'_j| > \sigma$ for $j = 1, \ldots, r'$, whereas $|R'_j| \le \sigma$ for $j = r'+1, \ldots, r$, which implies

$$r' = \sum_{i=1}^{m'} r_i = \sum_{i=1}^{m'} t_i 2^{s_i} > \frac{4\varrho\,(1+\varepsilon'^2)}{\varepsilon'^2} \sum_{i=1}^{m'} t_i \tag{15}$$

according to (8), (12)–(13), and (11). We include the remaining constant-size classes R'_j for $j = r'+1, \ldots, r$ into Q, that is,

$$Q' = Q \cup \bigcup_{j=r'+1}^{r} R'_j \tag{16}$$

whose size can be upper bounded as

$$|Q'| \le \log n + \sum_{i=m'+1}^{m} r_i \log\left(\frac{4\varrho\,(1+\varepsilon'^2)}{\varepsilon'^2}\right) < 2\log n \tag{17}$$

for sufficiently large n, since

$$\sum_{i=m'+1}^{m} r_i = \sum_{i=m'+1}^{m} t_i 2^{s_i} < \frac{4\varrho\,(1+\varepsilon'^2)}{\varepsilon'^2} \ln\frac{1}{\varepsilon'} \tag{18}$$

according to (8), (10), (14), and (11). This completes the definition of new classes $Q', R'_1, \ldots, R'_{r'}$. In addition, we define $\mathbf{c}' \in \{0,1\}^n$ that differs from $\mathbf{c}$ exactly on the constant number of bit locations from $R'_{r'+1}, \ldots, R'_r$, e.g.

$$c'_i = \begin{cases} 1 - c_i & \text{if } i \in \bigcup_{j=r'+1}^{r} R'_j \\ c_i & \text{otherwise.} \end{cases} \tag{19}$$

The modified $Q', R'_1, \ldots, R'_{r'}$ and $\mathbf{c}'$ are used in the following lemma for lower bounding the probability (5).

Lemma 1

$$p \ge \frac{\left|\mathcal{A}_n^{Q'}(\mathbf{c}') \setminus \bigcup_{j=1}^{r'} \mathcal{A}_n^{R'_j}(\mathbf{c}')\right|}{|\mathcal{A}_n|} = \frac{\left|\mathcal{A}_n^{Q'}(\mathbf{c}')\right|}{|\mathcal{A}_n|} - \frac{\left|\bigcup_{j=1}^{r'} \mathcal{A}_n^{R'_j \cup Q'}(\mathbf{c}')\right|}{|\mathcal{A}_n|}\,. \tag{20}$$

Proof. For verifying the lower bound in (20) it suffices to show that $\mathcal{A}_n^{Q'}(\mathbf{c}') \setminus \bigcup_{j=1}^{r'} \mathcal{A}_n^{R'_j}(\mathbf{c}') \subseteq \mathcal{A}_n^{Q}(\mathbf{c}) \setminus \bigcup_{j=1}^{r} \mathcal{A}_n^{R_j}(\mathbf{c})$ according to (5). Assume $\mathbf{a} \in \mathcal{A}_n^{Q'}(\mathbf{c}') \setminus \bigcup_{j=1}^{r'} \mathcal{A}_n^{R'_j}(\mathbf{c}')$, which means $\mathbf{a} \in \mathcal{A}_n^{Q'}(\mathbf{c}') \subseteq \mathcal{A}_n^{Q}(\mathbf{c}') = \mathcal{A}_n^{Q}(\mathbf{c})$ and $\mathbf{a} \notin \mathcal{A}_n^{R'_j}(\mathbf{c}') = \mathcal{A}_n^{R'_j}(\mathbf{c}) \supseteq \mathcal{A}_n^{R_j}(\mathbf{c})$ for every $j = 1, \ldots, r'$ by definitions (6), (16), (19), and the fact that $S_1 \subseteq S_2$ implies $\mathcal{A}_n^{S_2}(\mathbf{c}) \subseteq \mathcal{A}_n^{S_1}(\mathbf{c})$. In addition, $\mathbf{a} \in \mathcal{A}_n^{Q'}(\mathbf{c}')$ implies $\mathbf{a} \notin \mathcal{A}_n^{R_j}(\mathbf{c})$ for every $j = r'+1, \ldots, r$ according to (19), and hence, $\mathbf{a} \in \mathcal{A}_n^{Q}(\mathbf{c}) \setminus \bigcup_{j=1}^{r} \mathcal{A}_n^{R_j}(\mathbf{c})$. This completes the proof of the lower bound, while the equality in (20) follows from $\mathcal{A}_n^{R'_j \cup Q'}(\mathbf{c}') \subseteq \mathcal{A}_n^{Q'}(\mathbf{c}')$ for every $j = 1, \ldots, r'$. □

5 Almost k-Wise Independence

Furthermore, we will upper bound the probability of the finite union of events appearing in formula (20) by using Bonferroni inequality for constant number $C' = \min(C, r')$ of terms, which gives

$$p \geq \frac{\left|\mathcal{A}_n^{Q'}(\mathbf{c}')\right|}{|\mathcal{A}_n|} - \sum_{k=1}^{C'} (-1)^{k+1} \sum_{1 \leq j_1 < j_2 < \cdots < j_k \leq r'} \frac{\left|\bigcap_{i=1}^{k} \mathcal{A}_n^{R'_{j_i} \cup Q'}(\mathbf{c}')\right|}{|\mathcal{A}_n|} \tag{21}$$

$$= \sum_{k=0}^{C'} (-1)^k \sum_{1 \leq j_1 < j_2 < \cdots < j_k \leq r'} \frac{\left|\mathcal{A}_n^{\bigcup_{i=1}^{k} R'_{j_i} \cup Q'}(\mathbf{c}')\right|}{|\mathcal{A}_n|} \tag{22}$$

according to Lemma 1. For notational simplicity, the inner sum in (22) over $1 \leq j_1 < j_2 < \cdots < j_k \leq r'$ for $k = 0$ reads formally as it includes one summand $|\mathcal{A}_n^{Q'}(\mathbf{c}')|/|\mathcal{A}_n|$. Note that C' is odd for $C < r'$, while equality holds in (21) for $C' = r'$, which is the probabilistic inclusion-exclusion principle. For any $0 \leq k \leq C' \leq C$, we know $\left|\bigcup_{i=1}^{k} R'_{j_i} \cup Q'\right| \leq \lceil (C+2) \log n \rceil$ according to (6) and (17), and hence,

$$\frac{\left|\mathcal{A}_n^{\bigcup_{i=1}^{k} R'_{j_i} \cup Q'}(\mathbf{c}')\right|}{|\mathcal{A}_n|} \geq \frac{1}{2^{|Q'| + \sum_{i=1}^{k} \left|R'_{j_i}\right|}} - \beta = \frac{1}{2^{|Q'|}} \prod_{i=1}^{k} \frac{1}{2^{\left|R'_{j_i}\right|}} - \beta \tag{23}$$

(where the product in (23) equals formally 1 for $k = 0$) and similarly,

$$-\frac{\left|\mathcal{A}_n^{\bigcup_{i=1}^{k} R'_{j_i} \cup Q'}(\mathbf{c}')\right|}{|\mathcal{A}_n|} \geq -\frac{1}{2^{|Q'|}} \prod_{i=1}^{k} \frac{1}{2^{\left|R'_{j_i}\right|}} - \beta \tag{24}$$

according to (4) since $\mathcal{A}$ is $(\lceil (C+2) \log n \rceil, \beta)$-wise independent. We plug these inequalities into (22), which leads to

$$p \geq \sum_{k=0}^{C'} (-1)^k \sum_{1 \leq j_1 < j_2 < \cdots < j_k \leq r'} \frac{1}{2^{|Q'|}} \prod_{i=1}^{k} \frac{1}{2^{\left|R'_{j_i}\right|}} - \beta \sum_{k=0}^{C'} \binom{r'}{k}$$

$$\geq \frac{1}{2^{|Q'|}} \left(\sum_{k=0}^{C'} (-1)^k \sum_{1 \leq j_1 < j_2 < \cdots < j_k \leq r'} \prod_{i=1}^{k} \frac{1}{2^{\left|R'_{j_i}\right|}} - \beta\, 2^{|Q'|} (r'+1)^{C'} \right), \tag{25}$$

where

$$\beta\, 2^{|Q'|} (r'+1)^{C'} < \frac{1}{n^{C+3}}\, n^2\, n^C = \frac{1}{n} < \frac{\varepsilon'}{8} \tag{26}$$

for sufficiently large $n > 8/\varepsilon'$ by using the assumption on β, inequality (17), $r' < n$ (e.g., $r' = n$ would break (11)–(13)), and $C' \leq C$. The following lemma rewrites the inner sum in formula (25).

Lemma 2. *For* $0 \le k \le C'$,

$$\sum_{1\le j_1<j_2<\cdots<j_k\le r'} \prod_{i=1}^{k} \frac{1}{2^{|R'_{j_i}|}} = \sum_{\substack{k_1+\cdots+k_{m'}=k\\ 0\le k_1\le r_1,\ldots,0\le k_{m'}\le r_{m'}}} \prod_{i=1}^{m'} \frac{t_i^{k_i}}{k_i!} \prod_{j=1}^{k_i-1}\left(1-\frac{j}{r_i}\right) . \quad (27)$$

Proof. By grouping the classes of the same cardinality together, the left-hand side of inequality (27) can be rewritten as

$$\sum_{1\le j_1<j_2<\cdots<j_k\le r'} \prod_{i=1}^{k} \frac{1}{2^{|R'_{j_i}|}} = \sum_{\substack{k_1+k_2+\cdots+k_{m'}=k\\ 0\le k_1\le r_1,\ldots,0\le k_{m'}\le r_{m'}}} \prod_{i=1}^{m'} \binom{r_i}{k_i}\left(\frac{1}{2^{s_i}}\right)^{k_i} , \quad (28)$$

where $k_1,\ldots,k_{m'}$ denote the numbers of classes of corresponding cardinalities $s_1,\ldots,s_{m'}$ considered in a current summand, and

$$\binom{r_i}{k_i}\left(\frac{1}{2^{s_i}}\right)^{k_i} = \frac{r_i\,(r_i-1)\cdots(r_i-k_i+1)}{k_i!}\left(\frac{t_i}{r_i}\right)^{k_i} = \frac{t_i^{k_i}}{k_i!}\prod_{j=1}^{k_i-1}\left(1-\frac{j}{r_i}\right) \quad (29)$$

according to (8). □

Thus, we plug equations (26) and (27) into (25) and obtain

$$p > \frac{1}{n^2}\left(\sum_{k=0}^{C'}(-1)^k \sum_{\substack{k_1+\cdots+k_{m'}=k\\ 0\le k_1\le r_1,\ldots,0\le k_{m'}\le r_{m'}}} \prod_{i=1}^{m'} \frac{t_i^{k_i}}{k_i!}\prod_{j=1}^{k_i-1}\left(1-\frac{j}{r_i}\right) - \frac{\varepsilon'}{8}\right) . \quad (30)$$

Note that for $m'=0$ (implying $r'=C'=0$), the inner sum in (30) equals 1.

6 Frequent Cardinalities

We sort out the terms with frequent cardinalities (12) from the sum in formula (30), that is,

$$p > \frac{1}{n^2}\left(\sum_{k=0}^{C'}(-1)^k \sum_{\substack{k_1+\cdots+k_{m''}=k\\ 0\le k_1\le r_1,\ldots,0\le k_{m''}\le r_{m''}}} \prod_{i=1}^{m''} \frac{t_i^{k_i}}{k_i!}\prod_{j=1}^{k_i-1}\left(1-\frac{j}{r_i}\right) - T_1 - \frac{\varepsilon'}{8}\right) , \quad (31)$$

where the inner sum in (31) equals zero for $k > r'' = \sum_{i=1}^{m''} r_i$, and

$$T_1 = \sum_{k=0}^{C'}(-1)^{k+1} \sum_{\substack{k_1+\cdots+k_{m'}=k\\ 0\le k_1\le r_1,\ldots,0\le k_{m'}\le r_{m'}\\ (\exists\, m''+1\le \ell\le m')\; k_\ell>0}} \prod_{i=1}^{m'} \frac{t_i^{k_i}}{k_i!}\prod_{j=1}^{k_i-1}\left(1-\frac{j}{r_i}\right) \quad (32)$$

sums up the terms including rare cardinalities (13). In addition, we know

$$1 \geq \prod_{i=1}^{m''} \prod_{j=1}^{k_i-1} \left(1 - \frac{j}{r_i}\right) > \left(1 - \frac{C}{\varrho}\right)^C = 1 - \frac{\varepsilon'^2}{4(1+\varepsilon'^2)} \tag{33}$$

according to (12), (11), and $k_i \leq k = \sum_{i=1}^{m''} k_i \leq C' \leq C < \varrho$. The upper and lower bound (33) on the underlying product are used to lower bound the negative terms of (31) for odd k and the positive terms for even k, respectively, that is,

$$p > \frac{1}{n^2}\left(\sum_{k=0}^{C'} (-1)^k \sum_{\substack{k_1+\cdots+k_{m''}=k \\ 0\leq k_1 \leq r_1,\ldots,0\leq k_{m''}\leq r_{m''}}} \prod_{i=1}^{m''} \frac{t_i^{k_i}}{k_i!} - \frac{\varepsilon'^2}{4(1+\varepsilon'^2)} T_2 - T_1 - \frac{\varepsilon'}{8}\right) \tag{34}$$

where

$$T_2 = \sum_{k=0,2,4,\ldots}^{C'} \sum_{\substack{k_1+\cdots+k_{m''}=k \\ 0\leq k_1 \leq r_1,\ldots,0\leq k_{m''}\leq r_{m''}}} \prod_{i=1}^{m''} \frac{t_i^{k_i}}{k_i!}\,. \tag{35}$$

The following lemma upper bounds the above-introduced terms T_1 and T_2.

Lemma 3

(i) $T_1 < \frac{\varepsilon'}{8}$.

(ii) $T_2 < \frac{1+\varepsilon'^2}{2\,\varepsilon'}$.

Proof.

(i) We can only take the terms of (32) for odd $k = 1, 3, 5, \ldots$ into account since those for even k are nonpositive (e.g. the term for $k = 0$ equals zero because there is no $m'' + 1 \leq \ell \leq m'$ such that $k_\ell > 0$ in this case). Thus,

$$\begin{aligned} T_1 &\leq \sum_{k=1,3,5,\ldots}^{C'} \sum_{\substack{k_1+\cdots+k_{m'}=k \\ 0\leq k_1 \leq r_1,\ldots,0\leq k_{m'}\leq r_{m'} \\ (\exists\, m''+1\leq \ell \leq m')\, k_\ell>0}} \frac{r_\ell}{2^{s_\ell}} \frac{1}{k_\ell} \frac{t_\ell^{k_\ell-1}}{(k_\ell-1)!} \prod_{\substack{i=1 \\ i\neq \ell}}^{m'} \frac{t_i^{k_i}}{k_i!} \\ &\leq \frac{\varrho}{2^\sigma} \sum_{k=1,3,5,\ldots}^{C'} \sum_{\substack{k_1+\cdots+k_{m'}=k \\ 0\leq k_1 \leq r_1,\ldots,0\leq k_{m'}\leq r_{m'} \\ (\exists\, m''+1\leq \ell \leq m')\, k_\ell>0}} \frac{t_\ell^{k_\ell-1}}{(k_\ell-1)!} \prod_{\substack{i=1 \\ i\neq \ell}}^{m'} \frac{t_i^{k_i}}{k_i!} \end{aligned} \tag{36}$$

according to (8) and (13). Formula (36) is rewritten by replacing indices $k_\ell - 1$ and $k - 1$ with k_ℓ and k, respectively, which is further upper bounded by removing the upper bounds that are set on indices $k_1, \ldots, k_{m'}$ and by omitting the condition concerning the existence of special index ℓ, as follows:

$$T_1 \leq \frac{\varrho}{2^\sigma} \sum_{k=0,2,4,\ldots}^{C'-1} \sum_{\substack{k_1+\cdots+k_{m'}=k \\ k_1\geq 0,\ldots,k_{m'}\geq 0}} \prod_{i=1}^{m'} \frac{t_i^{k_i}}{k_i!} = \frac{\varrho}{2^\sigma} \sum_{k=0,2,4,\ldots}^{C'-1} \frac{\left(\sum_{i=1}^{m'} t_i\right)^k}{k!}\,, \tag{37}$$

where the multinomial theorem is employed. Notice that the sum on the right-hand side of equation (37) represents the first few terms of Taylor series of the hyperbolic cosine at point $\sum_{i=1}^{m'} t_i \geq 0$, which implies

$$T_1 < \frac{\varrho}{2^\sigma} \cosh\left(\sum_{i=1}^{m'} t_i\right) < \frac{\varepsilon'^2}{4(1+\varepsilon'^2)} \cdot \frac{\frac{1}{\varepsilon'} + \varepsilon'}{2} = \frac{\varepsilon'}{8} \tag{38}$$

according to (10) and (11) since the hyperbolic cosine is an increasing function for nonnegative arguments.

(ii) Similarly as in the proof of (i), we apply the multinomial theorem (cf. (37)) and the Taylor series of the hyperbolic cosine (cf. (38)) to (35), which gives

$$T_2 \leq \sum_{k=0,2,4,\ldots}^{C'} \sum_{\substack{k_1+\cdots+k_{m''}=k \\ k_1\geq 0,\ldots,k_{m''}\geq 0}} \prod_{i=1}^{m''} \frac{t_i^{k_i}}{k_i!} \leq \cosh\left(\sum_{i=1}^{m''} t_i\right) < \frac{1+\varepsilon'^2}{2\,\varepsilon'} \,. \tag{39}$$

□

We plug the bounds from Lemma 3 into (34) and obtain

$$p > \frac{1}{n^2}\left(\sum_{k=0}^{C'} (-1)^k \sum_{\substack{k_1+\cdots+k_{m''}=k \\ 0\leq k_1\leq r_1,\ldots,0\leq k_{m''}\leq r_{m''}}} \prod_{i=1}^{m''} \frac{t_i^{k_i}}{k_i!} - \frac{3\,\varepsilon'}{8}\right) . \tag{40}$$

7 Taylor's Theorem

In order to apply the multinomial theorem again, we remove the upper bounds that are set on indices in the inner sum of formula (40), that is,

$$p > \frac{1}{n^2}\left(\sum_{k=0}^{C'} (-1)^k \sum_{\substack{k_1+\cdots+k_{m''}=k \\ k_1\geq 0,\ldots,k_{m''}\geq 0}} \prod_{i=1}^{m''} \frac{t_i^{k_i}}{k_i!} - T - \frac{3\,\varepsilon'}{8}\right) , \tag{41}$$

which is corrected by introducing additional term

$$T = \sum_{k=0}^{C'} (-1)^k \sum_{\substack{k_1+\cdots+k_{m''}=k \\ k_1\geq 0,\ldots,k_{m''}\geq 0 \\ (\exists 1\leq \ell\leq m'')\, k_\ell > r_\ell}} \prod_{i=1}^{m''} \frac{t_i^{k_i}}{k_i!} \,. \tag{42}$$

Thus, inequality (41) can be further rewritten as

$$p > \frac{1}{n^2}\left(\sum_{k=0}^{C'}\frac{\left(-\sum_{i=1}^{m''}t_i\right)^k}{k!} - T - \frac{3\,\varepsilon'}{8}\right) \tag{43}$$

$$= \frac{1}{n^2}\left(e^{-\sum_{i=1}^{m''}t_i} - \mathcal{R}_{C'+1}\left(-\sum_{i=1}^{m''}t_i\right) - T - \frac{3\,\varepsilon'}{8}\right), \tag{44}$$

where Taylor's theorem is employed for the exponential function at point $-\sum_{i=1}^{m''}t_i$ producing the Lagrange remainder

$$\mathcal{R}_{C'+1}\left(-\sum_{i=1}^{m''}t_i\right) = \frac{\left(-\sum_{i=1}^{m''}t_i\right)^{C'+1}}{(C'+1)!}\,e^{-\vartheta\sum_{i=1}^{m''}t_i} < \left(\frac{\sum_{i=1}^{m''}t_i}{\sqrt{C'}}\right)^{C'+1} \tag{45}$$

with parameter $0 < \vartheta < 1$. Note that the upper bound in (45) assumes $C' > 0$, whereas for $C' = r' = 0$ implying $m'' = m' = 0$, we know $\mathcal{R}_1(0) = 0$. This remainder together with term T are upper bounded in the following lemma.

Lemma 4
(i) $T < \frac{\varepsilon'}{8}$.
(ii) $\mathcal{R}_{C'+1}\left(-\sum_{i=1}^{m''}t_i\right) < \frac{\varepsilon'}{4}$.

Proof
(i) We take only the summands of (42) for even $k \geq 2$ into account since the summands for odd k are not positive, while for $k = 0$ there is no $1 \leq \ell \leq m''$ such that $0 = k \geq k_\ell > r_\ell \geq 1$, which gives

$$T \leq \sum_{k=2,4,6,\ldots}^{C'}\sum_{\substack{k_1+\cdots+k_{m''}=k\\k_1\geq 0,\ldots,k_{m''}\geq 0\\(\exists 1\leq\ell\leq m'')\,k_\ell>r_\ell}}\frac{1}{2^{s_\ell}}\,\frac{r_\ell}{k_\ell}\,\frac{t_\ell^{k_\ell-1}}{(k_\ell-1)!}\prod_{\substack{i=1\\i\neq\ell}}^{m''}\frac{t_i^{k_i}}{k_i!}$$

$$\leq \frac{1}{2^\sigma}\sum_{k=2,4,6,\ldots}^{C'}\sum_{\substack{k_1+\cdots+k_{m''}=k\\k_1\geq 0,\ldots,k_{m''}\geq 0\\(\exists 1\leq\ell\leq m'')\,k_\ell>r_\ell}}\frac{t_\ell^{k_\ell-1}}{(k_\ell-1)!}\prod_{\substack{i=1\\i\neq\ell}}^{m''}\frac{t_i^{k_i}}{k_i!} \tag{46}$$

using (8) and (12). Formula (46) is rewritten by replacing indices $k_\ell - 1$ and $k - 1$ with k_ℓ and k, respectively, which is further upper bounded by omitting the condition concerning the existence of special index ℓ, as follows:

$$T \leq \frac{1}{2^\sigma}\sum_{k=1,3,5,\ldots}^{C'-1}\sum_{\substack{k_1+\cdots+k_{m''}=k\\k_1\geq 0,\ldots,k_{m''}\geq 0}}\prod_{i=1}^{m''}\frac{t_i^{k_i}}{k_i!} = \frac{1}{2^\sigma}\sum_{k=1,3,5,\ldots}^{C'-1}\frac{\left(\sum_{i=1}^{m''}t_i\right)^k}{k!}, \tag{47}$$

where the multinomial theorem is employed. Notice that the sum on the right-hand side of equation (47) represents the first few terms of Taylor series of the hyperbolic sine at point $\sum_{i=1}^{m''} t_i$, which implies

$$T \leq \frac{1}{2^\sigma} \sinh \left(\sum_{i=1}^{m''} t_i \right) < \frac{\varepsilon'^2}{4\varrho\,(1+\varepsilon'^2)} \cdot \frac{\frac{1}{\varepsilon'} - \varepsilon'}{2} < \frac{\varepsilon'}{8} \tag{48}$$

according to (10) and (11) since the hyperbolic sine is an increasing function.
(ii) For $C' = C \geq 1$, Lagrange remainder (45) can further be upper bounded as

$$\mathcal{R}_{C'+1} \left(-\sum_{i=1}^{m''} t_i \right) < \left(\frac{\ln \frac{1}{\varepsilon'}}{\sqrt{C}} \right)^{C+1} < \left(\frac{\varepsilon'}{2} \right)^{C+1} < \frac{\varepsilon'}{4} \tag{49}$$

for sufficiently large n by using (10) and the definition of C, while for $C' = r' < C$, the underlying upper bound

$$\mathcal{R}_{C'+1} \left(-\sum_{i=1}^{m''} t_i \right) \leq \left(\frac{\sum_{i=1}^{m'} t_i}{\frac{4\varrho\,(1+\varepsilon'^2)}{\varepsilon'^2}} \right)^{\frac{r'+1}{2}} < \frac{\ln \frac{1}{\varepsilon'}}{\frac{4\varrho\,(1+\varepsilon'^2)}{\varepsilon'^2}} < \frac{\varepsilon'}{4} \tag{50}$$

can be obtained from (15) and (10). □

Finally, inequality (9) together with the upper bounds from Lemma 4 are plugged into (44), which leads to

$$p > \frac{\varepsilon'}{4n^2} = \frac{\varepsilon}{4n^2} \left(1 - \frac{1}{\log n} \right) > 0 \tag{51}$$

according to (7). Thus, we have proven that for any $\mathbf{c} \in \{0,1\}^n$ the probability that there is $\mathbf{a} \in \mathcal{A}_n$ satisfying the conjunction (3) for Q and partition $\{R_1, \ldots, R_r\}$ is strictly positive, which means such $\mathbf{a}$ does exist. This completes the proof that $\mathcal{A}$ is ε-rich. □

8 Conclusion

In the present paper, we have made an important step in the effort to construct hitting set generators for the model of read-once branching programs of bounded width. Such constructions have so far been known only in the case of width 2 and in very restricted cases of bounded width (e.g. permutation or regular oblivious read-once branching programs). We have now provided an explicit polynomial-time construction of a hitting set for read-once branching programs of width 3 with the acceptance probability greater than $\sqrt{12/13}$. From the point of view of derandomization of unrestricted models, our result still appears to be unsatisfactory. The issue of whether our technique based on the richness condition can be extended to the case of width 4 or to bounded width represents an open problem for further research. Another challenge for improving our result is to optimize parameter ε, e.g. to achieve the result for $\varepsilon \leq \frac{1}{n}$, which would be important for practical derandomizations.

References

1. Alon, N., Goldreich, O., Håstad, J., Peralta, R.: Simple constructions of almost k-wise independent random variables. Random Structures and Algorithms 3(3), 289–304 (1992)
2. Bogdanov, A., Dvir, Z., Verbin, E., Yehudayoff, A.: Pseudorandomness for width 2 branching programs. ECCC Report No.70 (2009)
3. Braverman, M., Rao, A., Raz, R., Yehudayoff, A.: Pseudorandom generators for regular branching programs. In: Proceedings of the FOCS 2010 Fifty-First Annual IEEE Symposium on Foundations of Computer Science, pp. 41–50 (2010)
4. Brody, J., Verbin, E.: The coin problem, and pseudorandomness for branching programs. In: Proceedings of the FOCS 2010 Fifty-First Annual IEEE Symposium on Foundations of Computer Science, pp. 30–39 (2010)
5. De, A., Etesami, O., Trevisan, L., Tulsiani, M.: Improved pseudorandom generators for depth 2 circuits. In: Serna, M., Shaltiel, R., Jansen, K., Rolim, J. (eds.) APPROX 2010, LNCS, vol. 6302, pp. 504–517. Springer, Heidelberg (2010)
6. Goldreich, O., Wigderson, A.: Improved derandomization of BPP using a hitting set generator. In: Hochbaum, D.S., Jansen, K., Rolim, J.D.P., Sinclair, A. (eds.) RANDOM 1999 and APPROX 1999. LNCS, vol. 1671, pp. 131–137. Springer, Heidelberg (1999)
7. Koucký, M., Nimbhorkar, P., Pudlák, P.: Pseudorandom generators for group products. To appear in Proceedings of the STOC 2011 Forty-Third ACM Symposium on Theory of Computing. ACM, New York (2011)
8. Meka, R., Zuckerman, D.: Pseudorandom generators for polynomial threshold functions. In: Proceedings of the STOC 2010 Forty-Second ACM Symposium on Theory of Computing, pp. 427–436. ACM, New York (2010)
9. Nisan, N.: Pseudorandom generators for space-bounded computation. Combinatorica 12(4), 449–461 (1992)
10. Nisan, N., Wigderson, A.: Hardness vs. randomness. Journal of Computer and System Sciences 49(2), 149–167 (1994)
11. Šíma, J., Žák, S.: A polynomial time constructible hitting set for restricted 1-branching programs of width 3. In: van Leeuwen, J., Italiano, G.F., van der Hoek, W., Meinel, C., Sack, H., Plášil, F. (eds.) SOFSEM 2007. LNCS, vol. 4362, pp. 522–531. Springer, Heidelberg (2007)
12. Šíma, J., Žák, S.: A polynomial time construction of a hitting set for read-once branching programs of width 3. ECCC Report No. 88 (2010)
13. Šíma, J., Žák, S.: A sufficient condition for sets hitting the class of read-once branching programs of width 3 (submitted)
14. Wegener, I.: Branching Programs and Binary Decision Diagrams—Theory and Applications. SIAM Monographs on Discrete Mathematics and Its Applications. SIAM, Philadelphia (2000)

The Complexity of Inversion of Explicit Goldreich's Function by DPLL Algorithms

Dmitry Itsykson[1,⋆] and Dmitry Sokolov[2,⋆⋆]

[1] Steklov Institute of Mathematics at St. Petersburg
27 Fontanka, 191023, St. Petersburg, Russia
dmitrits@pdmi.ras.ru

[2] St. Petersburg Academic University
8(3) Khlopina, 194021, St.-Petersburg, Russia
sokolov.dmt@gmail.com

Abstract. The Goldreich's function has n binary inputs and n binary outputs. Every output depends on d inputs and is computed from them by the fixed predicate of arity d. Every Goldreich's function is defined by it's dependency graph G and predicate P. In 2000 O. Goldreich formulated a conjecture that if G is an expander and P is a random predicate of arity d then the corresponding function is one way. In 2005 M. Alekhnovich, E. Hirsch and D. Itsykson proved the exponential lower bound on the complexity of inversion of Goldreich's function based on linear predicate and random graph by myopic DPLL agorithms. In 2009 J. Cook, O. Etesami, R. Miller, and L. Trevisan extended this result to nonliniar predicates (but for a slightly weaker definition of myopic algorithms). Recently D. Itsykson and independently R. Miller proved the lower bound for drunken DPLL algorithms that invert Goldreich's function with nonlinear P and random G. All above lower bounds are randomized.

The main contribution of this paper is the simpler proof of the exponential lower bound of the Goldreich's function inversion by myopic DPLL algorithms. A dependency graph in our construction may be based on an arbitrary expander, particulary it is possible to use an explicit expander; the predicate may be linear or slightly nonlinear. Our definition of myopic algorithms is more general than one used by J. Cook et al. Our construction may be used in the proof of lower bound for drunken algorithms as well.

Keywords: DPLL algorithm, expander, one-way function, lower bounds.

⋆ Partially supported by Federal Target Programme "Scientific and scientific-pedagogical personnel of the innovative Russia" 2009-2013, RAS Program for Fundamental Research, the president grants NSh-5282.2010.1 and MK-4089.2010.1 and by RFBR.

⋆⋆ Partially supported by Federal Target Programme "Scientific and scientific-pedagogical personnel of the innovative Russia" 2009-2013 and by Yandex Fellowship.

1 Introduction

This work continues [1], [2], [3], [4] and is devoted to lower bounds of DPLL (for Davis, Putnam, Logemann, and Loveland) algorithms on satisfiable formulas. DPLL algorithm is a recursive algorithm. On each recursive call it simplifies an input formula F (without affecting its satisfiability), chooses a variable v and makes two recursive calls on the formulas $F[v := 1]$ and $F[v := 0]$ in some order. It returns the result "Satisfiable" if at least one of recursive calls returns "Satisfiable" (note that it is not necessary to make the second call if the first one was successful). Recursion stops if the input formula becomes trivial. That is, the algorithm is only allowed to backtrack when unsatisfiability in the current branch is proved. A DPLL algorithm is defined by simplification rules and two heuristics: the heuristic **A** chooses a variable for splitting and the heuristic **B** chooses a value that will be investigated first.

The behaivior of DPLL algorithms on unsatisfiable formulas is equivalent to tree-like resolution proofs. Therefore lower bounds on DPLL algorithms on unsatisfiable formulas follow from lower bound for resolutions [5]. However the most interesting inputs are satisfiable formulas. Consider for example formulas that code the problem of inversion of one-way function. The most important case for practice is the case there one-way function indeed has preimage. There is no hope of proving a superpolynomial lower bound for all DPLL algorithms on satisfiable formulas since if P = NP, then the heuristic that chooses the value of a variable that would be investigated first may always choose the correct value.

Exponential lower bounds on running time of myopic and drunken DPLL algorithms on satisfiable formulas were proved in the paper [1]; these two classes of DPLL algorithms cover a lot of known DPLL algorithms. In myopic algorithms heuristics that choose a variable for splitting and that choose a value that will be investigated first have the following restrictions: they can see the formula with erased signs of negations and they also know the number of positive and negative occurrences of every variable and they also can request $K = n^{1-\varepsilon}$ clauses of the formula to read them precisely. In drunken algorithms the heuristic that chooses variable for splitting may be arbitrary, while the first substituted value is chosen at random with equal probabilities. Lower bounds for myopic algorithms were proved on the formulas that code the system of linear equations over $\mathbb{F}_2$ based on expander matrices; lower bounds for drunken algorithms were proved on artificial formulas that are based on hard examples for resolution.

The paper [2] gives a cryptographic view on [1]. Namely it was noted in [2] that the lower bound for myopic algorithms [1] was proved on the formulas that code the problem of inversion of Goldreich's function based on linear predicate. Goldreich's function [6] has n binary inputs and n binary outputs. Every output depends on d inputs and is computed from them by a fixed predicate of arity d. Goldreich conjectured that if the dependency graph is an expander and the predicate is random, then the resulting function is one-way. However, linear functions are not interesting from the cryptographic point of view since they can be easily inverted by Gaussian elimination. The main goal of [2] was the proof of lower bound for a function that is potentially hard to invert. J. Cook et al.

consider Goldreich's function based on the predicate $x_1+x_2+\cdots+x_{d-2}+x_{d-1}x_d$ and a random graph (a random graph is an expander with high probability). They have proved the exponential lower bound for the weakened[1] variant of myopic algorithms. Recently Itsykson [3] and Miller [4] independently proved the lower bound on the complexity of inversion of Goldreich's function based on random graph and predicate of type $x_1 + x_2 + \cdots + x_{d-k} + Q(x_{d-k+1}, \ldots x_d)$, where Q is an arbitrary predicate of arity k and $k < d/4$ by drunken algorithms. We should note that the proof from [2] works for this type of predicates as well.

The construction of Goldreich's function in all papers listed above was randomized. In this paper we suggest an explicit construction of Goldreich's function based on expanders (for example the explicit expander from [7] fits our purposes). It is possible to use those formulas in the proof of exponential lower bound for drunken algorithms from [3]. In this paper we demonstrate the lower bound for myopic algorithms. Our proof is technically much simpler than proofs from [1] and [2]. We prove lower bound for the general notion of myopic algorithms (according to the definition from [1]) instead of the weakened variant that was used in [2].

Our Goldreich's function has the following structure: it is the sum of two Goldreich's functions: linear and nonlinear. The linear part is necessary for proving the lower bound for DPLL algorithms while the nonlinear part makes our function hard to invert in practice. The linear part is based on an expander, while nonlinear part may be almost arbitrary but it should depend only on $n^{\varepsilon/2}$ variables. Of course an adversary may guess the value of variables from nonlinear part and solve the resulting linear system by Gaussian elimination but the running time of such algorithm is $2^{n^{\varepsilon/2}}$ (still exponential), therefore we believe that there are hard invertible functions among our functions. We actually do not use in the proof the fact that the nonlinear part of predicate is the same for every bit of the output.

The plan of the proof is the following: first of all we slightly modify the expander from the linear part so that its adjacency matrix would have high rank. Since the nonlinear part of our function depends on very few variables we conclude that our Goldreich's function is almost a bijection. In order to prove the lower bound we first of all prove the lower bound for unsatisfiable formulas using lower bound techniques for resolutions from [8]. Using almost linearity and almost bijectivity we prove that with high probability the myopic algorithm makes the formula unsatisfiable during first several steps and we apply the lower bound for unsatisfiable formulas.

Our proof has one disadvantage compared to the proof from [1]; namely our proof works for expanders with degrees, that are large enough, while the technique from [1] works for degrees, that are at least 3. However the proof from [1] of the fact that a myopic algorithm with high probability makes the formula unsatisfiable during first several steps is complicated, while our proof is intuitive

[1] In contrast to [1], [2] did not allow the DPLL algorithm to use pure literal simplification rules and also myopic algorithms from [2] may read only constant (opposite to $n^{1-\varepsilon}$) number of clauses per step.

and based on the simple fact from elementary linear algebra: a dimension of a solution space of a satisfiable linear system does not depend on the right hand side.

2 Preliminaries

Let $X = \{x_1, x_2, \ldots, x_n\}$ be the set of propositional variables.

A partial substitution is a function $\rho : X \to \{0, 1, *\}$, that maps a variable to its value or leaves it free. The set $Vars(\rho) = \rho^{-1}(\{0, 1\})$ is the support of the substitution; we denote $|\rho| = |Vars(\rho)|$.

If ρ_1 and ρ_2 are two partial substitutions with disjoint support then the substitution $\rho_1 \cup \rho_2$ can be defined by the natural way.

We say that a string $y \in \{0, 1\}^n$ is consistent with the partial substitution ρ (we denote it $y \sim \rho$) if for all x_j from the support of ρ the following is satisfied $y_j = \rho(x_j)$.

2.1 DPLL Algorithms

We consider a wide class of SAT algorithms: DPLL (or backtracking) algorithms. A DPLL algorithm is defined by two *heuristics* (procedures): 1) Procedure **A** maps a CNF formula to one of its variables. (This is the variable for splitting). 2) Procedure **B** maps a CNF formula and its variable to $\{0, 1\}$. (This value will be investigated at first).

An algorithm may also use some syntactic *simplification rules*. Simplification rules may modify the formula without affecting its satisfiability and may also make substitutions to its variables if their values can be inferred from the satisfiability of the initial formula.

A DPLL algorithm is a recursive algorithm. Its input is a formula φ and a partial substitution ρ.

Algorithm 1. *Input: formula φ and substitution ρ*

- *Simplify φ by means of simplification rules (assume that simplification rules change φ and ρ; all variables that are substituted by ρ should be deleted from φ).*
- *If current formula is empty (that is, all its clauses are satisfied by ρ), then return ρ. If formula contains an empty clause (unsatisfiable), then return "formula is unsatisfiable".*
- *$x_j := \mathbf{A}(\varphi)$; $c := \mathbf{B}(\varphi, x_j)$*
- *Make a recursive call with the input $(\varphi[x_j := c], \rho \cup \{x_j := c\})$, if the result is "formula is unsatisfiable", then make a recursive call with the input $(\varphi[x_j := 1 - c], \rho \cup \{x_j := 1 - c\})$ and return its result, otherwise return the result of the first recursive call.*

Definition 1. *Myopic algorithms [1] are DPLL algorithms, where heuristics* **A** *and* **B** *have the following restrictions:*

– *They can see the whole formula with erased signs of negations.*
– *For every variable they know the number of its positive and the number of its negative occurrences.*
– *They may request to read $K = o(n)$ clauses to read precisely (with negation signs).*

Simplification rules: 1) Unit clause elimination*: if formula contains a clause with only one literal, then make a substitution that satisfies that clause. 2)* Pure literals rule*: if formula contains a variable that has only positive or only negative occurrences, then substitute it with the corresponding value.*

The running time of a DPLL algorithm for a given sequence of random bits is the number of recursive calls.

2.2 Expanders

We consider bipartite graphs with each part containing n vertices. The first part we denote by $X = \{x_1, x_2, \ldots, x_n\}$ and the second we denote by $Y = \{y_1, y_2, \ldots, y_n\}$. Every vertex from the set Y has an ordered list of its neighbours from the set X (repetitions are allowed). All considered graphs are d-regular: the degree of every vertex from Y is equal to d, where d is a constant.

Every graph has its adjacency matrix over $\mathbb{F}_2$. Rows of this matrix correspond to the set Y and columns correspond to the set X, the element with coordinates (y, x) contains the parity of the number of edges between y and x.

For set $A \subseteq Y$ we denote $\Gamma(A)$ (the set of neighbours of A) the set of vertices from X that are connected with at least one vertex from A; we denote $\delta(A)$ (the boundary of A) the set of vertices from X that have exactly one incoming edge from the set A.

Definition 2. *The graph G is a (r, d, c)-expander, if 1) the degree of any vertex in Y is equal to d; 2) for any set $A \subseteq Y, |A| \leq r$ we have $\Gamma(A) \geq c|A|$. The graph G is called a (r, d, c)-boundary expander if the second condition is replaced by: 2) for any set $A \subseteq Y, |A| \leq r$ we have $\delta(A) \geq c|A|$.*

Lemma 1 (cf. [1], Lemma 1). *Every (r, d, c)-expander is also a $(r, d, 2c - d)$-boundary expander.*

Proof. Let $A \subseteq Y$, $|A| \leq r$, then $|\Gamma(A)| \geq c|A|$. The number of edges between A and $\Gamma(A)$ may be estimated: $d|A| \geq |\delta(A)| + 2|\Gamma(A) \setminus \delta(A)| = 2|\Gamma(A)| - |\delta(A)| \geq 2c|A| - \delta(A)$. Finally we get $\delta(A) \geq (2c - d)|A|$. □

We need boundary expanders; for this it is enough to have an expander with constant $c > d/2$. For example, a random graph is an appropriate expander.

Lemma 2 ([9], Lemma 1.9). *For $d \geq 32$, for all big enough n a random bipartite d-regular graph, where parts X and Y contain n vertices is a $(\frac{n}{10d}, d, \frac{5}{8}d)$-expander with probability 0.9, if for every vertex in Y d edges are chosen independently at random (with repetitions).*

Corollary 1. *In terms of Lemma 2 this graph is a* $(\frac{n}{10d}, d, \frac{1}{4}d)$*-boundary expander.*

Proof. Follows from Lemma 1. $\square$

There are also explicit constructions of such expanders:

Lemma 3 ([7]). *For every constant* $\epsilon > 0$ *there is a constant* d *such that it is possible to construct a* (r, d, c)*-expander in polynomial of* n *time, where* $c = (1 - \epsilon)d$, $r = \Omega(n/d)$.

Corollary 2. *This graph is a* $(\Omega(n/d), d, (1 - 2\epsilon)d)$*-boundary expander.*

2.3 Goldreich's Function

O. Goldreich in the paper [6] introduces a function $f : \{0,1\}^n \to \{0,1\}^n$ defined by a graph G and a predicate $P : \{0,1\}^d \to \{0,1\}$. Every string from $\{0,1\}^n$ assignes some value to the variables from the set $X = \{x_1, x_2, \ldots, x_n\}$. The value of $(f(x))_j$ (j-th symbol of the string $f(x)$) is computed in the following way: if y_j has neighbours $x_{j_1}, x_{j_2}, \ldots, x_{j_d}$,then $(f(x))_j = P(x_{j_1}, x_{j_2}, \ldots, x_{j_d})$.

2.4 Formulas from Goldreich's Function

Now we describe the way we code the problem of inversion of Goldreich's function as instance of CNF satisfiability problem.

Let $g : \{0,1\}^\ell \to \{0,1\}$, the canonical CNF representation of g is the following: for every $c \in \{0,1\}^\ell$ that satisfies $g(c) = 0$ we write the clause $x_1^{c_1} \vee x_2^{c_2} \vee \cdots \vee x_\ell^{c_\ell}$, where $x_i^0 = x_i$ and $x_i^1 = \neg x_i$. The whole formula is the conjunction of all written clauses.

Let f be the Goldreich's function based on the graph G and the predicate P. We represent the equation $f(x) = b$ in the following way: for every vertex $y_j \in Y$ that has neighbours $x_{j1}, x_{j2}, \ldots, x_{jd}$ we put down the canonical CNF representation of the equality $b_j = P(x_{j1}, x_{j2}, \ldots, x_{jd})$ using variables $x_{j1}, x_{j2}, \ldots, x_{jd}$. The conjunctions of all those formulas we denote $\Phi_{f(x)=b}$. The part of this formula that corresponds to the vertices from the set $A \subseteq Y$ we denote $\Phi^A_{f(x)=b}$.

Lemma 4. *If a function* $g : \{0,1\}^\ell \to \{0,1\}$ *is linear on at least two variables, then the canonical CNF representation of* g *has exactly* $2^{\ell-1}$ *clauses and every variable has an equal number of positive and negative occurrences.*

Proof. Let g have the following $\mathbb{F}_2$ representation $g(x_1, x_2, \ldots, x_n) = x_1 + x_2 + h(x_3, \ldots, x_\ell)$. Let us denote $T_0 = h^{-1}(0)$, $T_1 = h^{-1}(1)$. Then $g^{-1}(0) = \{00y \mid y \in T_0\} \cup \{11y \mid y \in T_0\} \cup \{01x \mid y \in T_1\} \cup \{10y \mid y \in T_1\}$. The latter shows that $|g^{-1}(0)| = 2^{l-1}$ and every variable has an equal number of positive and negative occurrences. $\square$

Lemma 4 implies that if a myopic algorithm does not see negation signs, then it can't differ $g(x) = 0$ from $g(x) = 1$ when g is linear on at least 2 variables. Also we note that a canonical CNF formula is still canonical after substitution of the value of a variable.

3 Almost Bijective Goldreich's Function

3.1 Linear Function

Let G_1 be a d_1-regular graph and G_2 be a d_2-regular graph. $G_1 + G_2$ is $(d_1 + d_2)$-regular graph such that for every vertex from Y the list of neighbours is a concatenation of lists of neighbours in graph G_1 and graph G_2. The adjacency matrix of $G_1 + G_2$ is the sum of adjacency matrices of G_1 and G_2.

Proposition 1. *If graph G is a (r, d, c)-expander and G' is a d'-regular graph, then $G + G'$ is a $(r, d + d', c)$-expander.*

Theorem 1. *Given a graph G it is possible to construct in polynomial of n time a 1-regular graph T such that the rank of adjacency matrix of $G + T$ is at least $n - 1$.*

Proof. First of all we prove the auxiliary lemma:

Lemma 5. *Let $a = (\alpha_1, \ldots, \alpha_n) \in \mathbb{F}_2^n$. Then there are at least $n - 1$ linear independent vectors among $b_i = (\alpha_1, \ldots, \alpha_{i-1}, \alpha_i + 1, \alpha_{i+1}, \ldots, \alpha_n)$, where $1 \leq i \leq n$.*

Proof. Let us consider the matrix A of size $n \times n$; all columns of A are equal to vector a. Vectors b_i are columns of the matrix $A + E$, where E is the identity matrix. Since the rank of the sum of matrices is less then or equal to the the sum of ranks we may conclude that $n = \operatorname{rk} E \leq \operatorname{rk}(A + E) + \operatorname{rk} A$. All columns of A are the same, hence $\operatorname{rk} A \leq 1$ and $\operatorname{rk}(A + E) \geq n - 1$. □

Now we describe the construction of graph T. We start from an empty set of edges and we will add one edge per step. On the i-th step for $1 \leq i \leq n - 1$ we add a neighbour to the vertex $y_i \in Y$ in such a way that the first i rows of $G + T$ are linearly independent. It can be done by the Lemma 5 (we apply the Lemma to the i-th row of the matrix of graph G). We add an arbitrary neighbour to vertex y_n. □

Corollary 3. *If G is a (r, d, c)-expander, then the graph $G + T$ from the theorem is a $(r, d + 1, c)$-expander and the Goldreich's function f based on $G + T$ and a linear predicate of arity $d + 1$ has the following property: for every $b \in \{0, 1\}^n$ the size of the set $f^{-1}(b)$ is at most 2.*

3.2 Slightly Nonlinear Goldreich's Function

Let $R \subseteq X$ be some subset of X. R-graph is a regular graph such that all vertices from $X \setminus R$ have degree 0.

Lemma 6. *Let G be a $(d - k)$-regular graph with an adjacency matrix of rank at least $n - 1$ and H be a k-regular R-graph. Let f be Goldreich's function based on $G + H$ and predicate $x_1 + x_2 + \cdots + x_{d-k} + Q(x_{d-k+1}, \ldots, x_d)$, where Q is an arbitrary predicate of arity k. Then for every $b \in \{0, 1\}^n$ the size of the set $f^{-1}(b)$ is at most $2^{|R|+1}$.*

Proof. We consider the system of equalities $f(x) = b$ and fix values of all variables from the set R. We get the linear system whose matrix equals to the matrix of graph G after removing the columns from the set R. The matrix of G has rank $n-1$, therefore the resulting system has at most two solutions. Hence the initial system $f(x) = b$ has at most $2^{|R|+1}$ solutions. □

4 Lower Bound on Unsatisfiable Formulas

We say that a variable is *sensitive* if by changing its value we change the value of the formula (for every assignment of values of other variables). (The boolean function that corresponds to the formula is linear on all its sensitive variables).

Theorem 2 ([3]). *Let f be a Goldreich's function based on G and P, where graph G is a (r, d, c)-boundary expander and predicate P contains at most k insensitive variables; ρ is a partial assignment to variables of X such that the formula $\Phi_{f(x)=b}|_\rho$ is unsatisfiable and for any set of vertices $A \subseteq Y$, $|A| < \frac{r}{2}$, the formula $\Phi^A_{f(x)=b}|_\rho$ is satisfiable. Then the running time of any DPLL algorithm (that does not use simplification rules) on the formula $\Phi_{f(x)=b}|_\rho$ is at least $2^{\frac{(c-k)r}{4}-|\rho|-d}$.*

5 Lower Bound on Satisfiable Formulas

5.1 Closure

Let graph G be a (r, d, c)-boundary expander. Let $0 < k < c-2$ and $P(x_1, \dots, x_d) = x_1 + \cdots + x_{d-k} + Q(x_{d-k+1}, \dots, x_d)$; Goldreich's function f is based on G and P.

The next technical definition formalize the following simple idea: suppose that the set J is removed from the part X of G and we want to remove a set I from Y (and also $\Gamma(I)$ from X) such that the resulting graph becomes a $(r/2, d, k+1)$-boundary expander. We construct such I step by step removing sets with small boundary from Y.

Definition 3. *Let $J \subseteq X$. The set of vertices $I \subseteq Y$ is called k-closure of the set J if there is a finite sequence of sets $I_1, I_2, \dots, I_m$ (we denote $C_\ell = \bigcup_{1 \le i \le \ell} I_i$, $C_0 = \emptyset$), such that the following properties are satisfied:*

- *$I_\ell \subseteq Y$ and $0 < |I_\ell| \le \frac{r}{2}$ for all $1 \le \ell \le m$;*
- *$I_i \cap I_j = \emptyset$ for all $1 \le i, j \le m$;*
- *$|\delta(I_\ell) \setminus (\Gamma(C_{\ell-1}) \cup J)| \le (1+k)|I_l|$; for all $1 \le \ell \le m$;*
- *for all $I' \subseteq Y \setminus C_m$ if $0 < |I'| \le \frac{r}{2}$, then $|\delta(I') \setminus (\Gamma(C_m) \cup J)| > (1+k)|I'|$;*
- *$I = C_m$.*

The set of all k-closures of the set J we denote as $Cl^k(J)$.

Lemma 7. *1. For every set $J \subseteq X$ there exists a k-closure. 2. Let $J_1 \subseteq J_2$, then for every $I_1 \in Cl^k(J_1)$ there exists $I_2 \in Cl^k(J_2)$ such that $I_1 \subseteq I_2$*

Lemma 8 ([1]). *Let $|J| < \frac{(c-k-1)r}{2}$, then for every set $I \in Cl^k(J)$ the inequality $|I| \leq (c-k-1)^{-1}|J|$ is satisfied*

Definition 4. *Let $f : \{0,1\}^n \to \{0,1\}^n$ be the Goldreich's function based on graph G and predicate P, $b \in \{0,1\}$. Partial substitution ρ is called locally consistent for the equation $f(x) = b$ if there exists a string $z \in \{0,1\}^n$ that is consistent to ρ and a set $I \in Cl^k(Vars(\rho))$ such that the equality $f(z)|_I = b|_I$ holds.*

Lemma 9 (cf. [1]). *If the partial substitution ρ is locally consistent for $f(x) = b$, then for all $Z \subseteq X$, $|Z| \leq \frac{r}{2}$ there exists a string $z \in \{0,1\}^n$ such that z is consistent with ρ and the equality $f(z)|_Z = b|_Z$ holds.*

Proof. Proof by contradiction. Consider the minimal $Z \subseteq Y$ such that $|Z| \leq \frac{r}{2}$ and for all z that are consistent to ρ the nonequality $f(z)|_Z \neq b|_Z$ holds. Let $I \in Cl^k(Vars(\rho))$ be from Definition 4. Partial substitution ρ is locally consistent therefore $Z \setminus I \neq \emptyset$.

By the definition of closure $|\delta(Z \setminus I) \setminus (\Gamma(I) \cup Vars(\rho))| > (k+1)|Z \setminus I|$, therefore there exists $y \in Z \setminus I$ such that at least $k+1$ boundary vertices of set Z (not from the support of ρ and not connected with I) are connected with y. The minimality of Z implies that there exists $z \in \{0,1\}^n$, such that $z \sim \rho$ and $f(z)|_{Z\setminus\{y\}} = b|_{Z\setminus\{y\}}$. It is possible also to satisfy the equation corresponding to vertex y by flipping the z-value of one of the boundary neighbours of vertex y. Therefore there exists $z' \in \{0,1\}^n$ that is consistent with ρ and $f(z')|_Z = b|_Z$. Contradiction. □

5.2 Clever Myopic Algorithm

We assume that the myopic algorithm runs on the formula $\Phi_{f(x)=b}$, where $f^{-1}(b) \neq \emptyset$. We describe the clever myopic algorithm. A clever myopic algorithm is allowed to read more clauses precisely (equivalently it may open more bits of b). Besides, the clever algorithm doesn't make substitutions that obviously lead to unsatisfiable formulas. It is not hard to see that it is enough to prove the lower bound for clever myopic algorithms; the lower bound for all myopic algorithms will follow.

Now we describe the behavior of clever myopic algorithms more formally. A clever algorithm has a current partial substitution ρ and a set $I \in Cl^k(Vars(\rho))$. At the beginning $\rho = \emptyset$, $I = \emptyset$. On each step the clever algorithm simplifies the formula (probably increases ρ and extends the set I to the element of $Cl^k(Vars(\rho))$.

If the clever algorithm requests a clause that corresponds to the vertex $y_j \in Y$ we say that the algorithm opens j-th bit of output. We assume that all clauses corresponding to $y_j \in Y$ may be read by a clever algorithm for free.

Consider the heuristic **A** that choose variable x for splitting. Let Z be the set of all open bits of output (in particular Z includes K bits that were open before x was choosen). The clever algorithm extends the set I to the element of $Cl^k(Vars(\rho) \cup \{x\})$. The set of open bits is increased: $Z := Z \cup I$. The clever algorithm chooses the value of variable x in order to make the part of formula that corresponds to Z satisfiable.

Lemma 10. *For every clever myopic algorithm A there exists another clever myopic algorithm B such that B does not use pure literal and unit clause elimination rules and the running time of algorithm B on the formula $\Phi_{f(x)=b}$ is bounded by polynomial on the running time of algorithm A.*

Proof. If the current predicate in the vertex $y \in Y$ (taking into account ρ) is linear on at least two variables then Lemma 4 implies that there are no pure literals in the formula that corresponds to y. So predicates in vertices that contain pure literals have at most one linear variable. All such vertices are contained in $I \in Cl^k(Vars(\rho))$, hence all corresponding bits of output are open and the algorithm may make a substitution to this pure literal by itself. Similarly, if formula contains a unit clause, then the corresponding vertex is in I and a clever algorithm may choose the correct substitution by itself. $\square$

In the following we assume that clever myopic algorithms do not use simplification rules.

Let us denote $N = \lfloor \frac{(c-k-1)r}{4dK} \rfloor$.

Lemma 11 (cf. [1]). *After N steps of any clever myopic algorithm the number of open bits is at most $\frac{r}{2}$.*

Proof. The number of open bits is at most $K\frac{(c-k-1)r}{4dK} + |Cl^k(Vars(\rho))|$, where ρ is the current substitution. By Lemma 8 $|Cl^k(Vars(\rho)| \leq \frac{|Vars(\rho)|}{c-k-1}$. Since $|Vars(\rho))| \leq \frac{(c-k-1)r}{4}$ we may conclude $K\frac{(c-k-1)r}{4dK} + |Cl^k(Z)| \leq \frac{(c-k-1)r}{4d} + \frac{r}{4} \leq \frac{r}{2}$ $\square$

Corollary 4. *During the first N steps a clever myopic algorithm does not backtrack (backtracking corresponds to a leaf of the splitting tree) and ρ is locally consistent.*

Proof. During N steps the number of open bits is at most $\frac{r}{2}$. We prove by induction that the current substitution is locally consistent. It is trivial for the beginning. Induction step follows from the fact that the value of the variable is chosen in such a way that $\Phi_{f(x)=b}|_I$ is satisfiable. This is possible by Lemma 9 and by induction hypothesis. $\square$

Our goal is to show that after N steps of a clever myopic algorithm the current formula will be unsatisfiable with high probability.

From this point we assume that graph G has the type $G_L + H$, where G_L is a $(d-k)$-regular and H is a k-regular R-graph; the rank of adjacency matrix of G_L is at least $n-1$. The Goldreich's function f based on G and P is linear on variables $X \setminus R$.

Lemma 12. *Let $b \in \{0,1\}^n$ and $J \subseteq X$. Let $y \in \{0,1\}^n$ and $Z \subseteq Y$, we define set $X_y = \{x \in \{0,1\}^n \mid \forall j \in (X \setminus J)\ x_j = y_j\}$ and set $S_y = \{x \in X_y \mid f(x)|_Z = b|_Z\}$. Then either $|S_y| \geq 2^{|J|-|Z|-|J \cap R|}$ or $|S_y| = 0$.*

Proof. We have to estimate the number of $x \in X_y$ that satisfies the system of equalities $f(x)|_Z = b|_Z$. If we fix the values for variables x_j for $j \in J \cap R$ then the system becomes linear over variables x_j for $j \in (J \setminus R)$. The rank of the system does not exceed $|Z|$ and the number of variables is at least $|J| - |J \cap R|$ (it is not necessary for all those variables to have explicit occurrences in the system). Thus if a solution exists then the dimension of the solution space is at least $|J| - |J \cap R| - |Z|$. Since our system is over field $\mathbb{F}_2$ the number of solutions is at least $2^{|J|-|Z|-|J\cap R|}$ even for fixed values of x_j, $j \in J \cap R$. □

Let Z be the set of open bits b in the equation $f(x) = b$, ρ be some partial substitution; we denote $C_{\rho,Z,b}$ the set of $x \in \{0,1\}^n$ that are consistent with ρ and satisfy $f(x)|_Z = b|_Z$. Formally $C_{\rho,Z,b} = \{x \mid f(x)|_Z = b|_Z, x \sim \rho\}$.

Lemma 13. *Let $Z \subseteq Y$, $|Z| < \frac{r}{2}$, $J \subseteq X$. Then for every two locally consistent substitutions ρ_1, ρ_2 with $Vars(\rho_1) = Vars(\rho_2) = J$ and for every $b \in \{0,1\}^n$ the following is satisfied: $\frac{|C_{\rho_1,Z,b}|}{|C_{\rho_2,Z,b}|} \le 2^{|R|}$.*

Proof.
$$\frac{|C_{\rho_1,Z,b}|}{|C_{\rho_2,Z,b}|} = \frac{\sum\limits_{\sigma} |C_{\rho_1\cup\sigma,Z,b}|}{\sum\limits_{\sigma} |C_{\rho_2\cup\sigma,Z,b}|},$$
where the sum in both cases is over partial substitutions σ with support $Vars(\sigma) = R \setminus J$.

We show that the size of the set $C_{\rho_i\cup\sigma,Z,b}$ is either 0 or some fixed value and not dependant on σ and $i \in \{1,2\}$.

The size of the set $C_{\rho_i\cup\sigma,Z,b}$ equals the number of solutions of the system of equations $f(x)|_Z = b|_Z$ if some bits of x are fixed by substitution $\rho_i \cup \sigma$. This fixation makes the system linear. Note that the rank of this system does not depend on substitutions ρ_i and σ (since ρ_i and σ influence only the column of constants in the system). Therefore, if such system has a solution then the number of solutions does not depend on i and σ.

Since the substitution ρ_i is locally consistent and $|Z| < \frac{r}{2}$, Lemma 9 implies that there exists such substitution σ_i with support $Vars(\sigma_i) = R \setminus J$ that $C_{\rho_i\cup\sigma_i,Z,b} \neq \emptyset$.
$$\frac{|C_{\rho_1,Z,b}|}{|C_{\rho_2,Z,b}|} \le \frac{2^{|R|}|C_{\rho_1\cup\sigma_1,Z,b}|}{|C_{\rho_2\cup\sigma_2,Z,b}|} = 2^{|R|}.$$ □

Theorem 3. *Assume $|R| = o(\frac{n}{K})$ and ρ is the current substitution after N steps of a clever myopic algorithm running on the fomula $\Phi_{f(x)=b}$ for some $b \in f(\{0,1\}^n)$ and Z is the set of open bits. Then $\Pr_{y\leftarrow U(\{0,1\}^n)}[\exists x : x \sim \rho,\ f(x) = f(y) \mid f(y)|_Z = b|_Z] \le 2^{-\Omega(\frac{n}{K})}$.*

Before giving a formal prove we informally describe the main idea. For simplicity we assume that $R = \emptyset$ therefore the predicate P is linear . We consider a clever myopic algorithm after N steps (i.e. $|\rho| = N$). In this moment the size of the set $I \in Cl^k(Vars(\rho))$ does not exceed $(1-\varepsilon)N$ for some positive ε by Lemma 8. We apply Lemma 12 for $J = Vars(\rho)$ and $Z = I$, Lemma 12 states that the number of locally consistent substitutions is at least $2^{|Vars(\rho)|-|I|} = 2^{\Omega(N)}$.

Lemma 9 and Lemma 11 imply that every local consistent partial substitution may be extended to the full substitution that is consistent with open bits of the right hand side. Lemma 13 states that the number of such extensions is the same for every locally consistent substitution if $R = \emptyset$. A myopic algorithm has no chance to find one substitution among all locally consistent substitutions since they all have equal chances to be correct. Since our linear system has at most two solutions (if $R = \emptyset$), there are at most two locally consistent substitutions that can be extended to the solution of the system. Therefore the probability of correct substitution is at most $2^{-\Omega(N)} = 2^{-\Omega(\frac{n}{K})}$.

Proof (Theorem 3). Corollary 4 implies that during N steps the algorithm does not backtrack and $|\rho| = N$.

We apply Lemma 12 for $J = Vars(\rho)$ and $Z = I$, where $I \in Cl^k(Vars(\rho))$ is from definition of a clever myopic algorithm after step N. Since $b \in f(\{0,1\}^n)$ there exists $y \in \{0,1\}^n$ such that $S_y \neq \emptyset$ (S_y is defined in the Lemma 12) and the inequality $|S_y| \geq 2^{|Vars(\rho)|-|I|-|R|}$ holds. Therefore at least $2^{|Vars(\rho)|-|I|-|R|}$ substitutions with support $Vars(\rho)$ are locally consistent.

$$\begin{aligned}
&\Pr_{y \leftarrow U(\{0,1\}^n)}[\exists x : x \sim \rho,\ f(x) = f(y) \mid f(y)|_Z = b|_Z] \\
&\qquad = \Pr_{y \leftarrow U(\{0,1\}^n)}[f^{-1}(f(y)) \cap C_{\rho,Z,b} \neq \emptyset \mid f(y)|_Z = b|_Z] \\
&\qquad\qquad \leq \max_y |f^{-1}(f(y))| \cdot \Pr_{y \leftarrow U(\{0,1\}^n)}[y \in C_{\rho,Z,b} \mid f(y)|_Z = b|_Z]
\end{aligned}$$

By Lemma 6 the first term may be estimated as $\max_y |f^{-1}(f(y))| \leq 2^{|R|+1}$. Let us estimate the second term: $\Pr_{y \leftarrow U(\{0,1\}^n)}[y \in C_{\rho,Z,b} \mid f(y)|_Z = b|_Z] \leq \frac{\max_\sigma |C_{\sigma,Z,b}|}{\sum_\sigma |C_{\sigma,Z,b}|}$, where σ goes through all locally correct substitutions with the support $Vars(\rho)$. By Lemma 13 $\frac{\max_\sigma |C_{\sigma,Z,b}|}{\sum_\sigma |C_{\sigma,Z,b}|} \leq 2^{|R|} \frac{\min_\sigma |C_{\sigma,Z,b}|}{2^{|Vars(\rho)|-|I|-|R|} \min_\sigma |C_{\sigma,Z,b}|} = 2^{2|R|+|I|-|Vars(\rho)|}$.

Altogether:

$$\Pr_{y \leftarrow U(\{0,1\}^n)}[\exists x : x \sim \rho,\ f(x) = f(y) \mid f(y)|_Z = b|_Z] \leq 2^{3|R|+|I|-|Vars(\rho)|+1}.$$

Since $I \in Cl^k(Vars(\rho))$ the Lemma 8 implies $|I| \leq (c-k-1)^{-1}|Vars(\rho)|$. The statement of the theorem follows from $Vars(\rho) = \Omega(\frac{n}{K})$ and $c > k+2$. □

Theorem 4. *Let $|R| = o(\frac{n}{K})$, then for every myopic algorithm A the following inequality holds:* $\Pr_{y,s}[t_A(\Phi_{f(x)=f(y)}) \geq 2^{\Omega(n)}] \geq 1 - 2^{-\Omega(\frac{n}{K})}$, *where $t_A(x)$ denotes the running time of A on input x and s is a string of random bits used by A.*

Proof. Lemma 10 implies that it is enough to prove the Theorem for clever myopic algorithms that do not use simplification rules.

We fix the string of random bits s and prove that for algorithms that use s instead of random bits the following holds: $\Pr_y[t_A(\Phi_{f(x)=f(y)}) \geq 2^{\Omega(n)}] \geq 1 - 2^{-\Omega(\frac{n}{K})}$, and the theorem follows.

We consider a clever myopic algorithm after N steps on the formula $\Phi_{f(x)=f(y)}$. Let Z_y be the set of open bits of output by this moment. Note that for a fixed string s the behavior of algorithm during the first N steps is the same for all $y' \in \{0,1\}^n$ such that $f(y')|_{Z_y} = f(y)|_{Z_y}$ (in this case $Z_{y'} = Z_y$). Thus the set of all $y \in \{0,1\}^n$ may be split on the finite number of classes of equivalence $S_1, S_2, \dots, S_m$ such that for all y and for all $y' \in S_y$ the values of Z'_y are the same and the values of $f(y)|_{Z_y}$ are the same, and this is not true for different classes.

$$\Pr_y[t_A(\Phi_{f(x)=f(y)}) \geq 2^{\Omega(n)}] = \sum_{i=1}^{m} \Pr_y[t_A(\Phi_{f(x)=f(y)}) \geq 2^{\Omega(n)} \mid y \in S_i] \Pr_y[y \in S_i].$$

By Theorem 3 after N steps of a clever myopic algorithm

$$\Pr_y[\Phi_{f(x)=f(y)}|_\rho \text{ is unsatisfiable } \mid y \in S_i] \geq 1 - 2^{-\Omega(\frac{n}{K})},$$

where ρ is the current substitution that is locally consistent. Finally Theorem 2 implies that $\Pr_y[t_A(\Phi_{f(x)=f(y)}) \geq 2^{\Omega(n)} \mid y \in S_i] \geq 1 - 2^{-\Omega(\frac{n}{K})}$. The theorem follows from the last inequality. □

In conclusion we describe the construction of Goldreich's function that suits the previous theorem.

We choose $\epsilon = \frac{1}{4k+1}$ and for given ϵ we construct an $(r, d, (1-\epsilon)c)$-expander H by Lemma 3. The constant d satisfies the inequality $d \geq 4k+1$. By Theorem 1 we add to the constructed graph such 1-regular graph T that the resulting graph $H+T$ has the adjacency matrix with rank at least $n-1$. The resulting graph is a $(r, d+1, (1-\frac{1}{4k+1})d)$-expander. We choose the subset $R \subseteq X$ of size $o(n/K)$ and k-regular R-graph F. We define $G = H + T + F$; graph G is a $(r, d+1+k, (1-\frac{1}{4k+1})d)$-expander and hence a $(r, d+1+k, d(1-\frac{1}{4k+1})-k-1)$-boundary expander. For $k > 1$ the inequality $d(1-\frac{1}{4k+1}) - k - 1 > k+2$ holds. For such graph any predicate of the type $x_1 + \dots + x_{d-k} + Q(x_{d_{k+1}}, \dots, x_d)$ is suitable, where Q is arbitrary predicate of arity k. It may be easily verified that we do not use the fact that the predicate Q is the same for all vertices of the set Y.

References

1. Alekhnovich, M., Hirsch, E.A., Itsykson, D.: Exponential lower bounds for the running time of DPLL algorithms on satisfiable formulas. J. Autom. Reason. 35(1-3), 51–72 (2005)
2. Cook, J., Etesami, O., Miller, R., Trevisan, L.: Goldreich's one-way function candidate and myopic backtracking algorithms. In: Proceedings of TCC, pp. 521–538. Springer, Heidelberg (2009)
3. Itsykson, D.: Lower bound on average-case complexity of inversion of goldreich's function by drunken backtracking algorithms. In: Ablayev, F., Mayr, E.W. (eds.) CSR 2010. LNCS, vol. 6072, pp. 204–215. Springer, Heidelberg (2010)

4. Miller, R.: Goldreich's one-way function candidate and drunken backtracking algorithms. Master's thesis, University of Virginia, Distinguished Majors Thesis (2009)
5. Tseitin, G.S.: On the complexity of derivation in the propositional calculus. Zapiski nauchnykh seminarov LOMI 8, 234–259 (1968); English translation of this volume: Consultants Bureau, N.Y., pp. 115–125 (1970)
6. Goldreich, O.: Candidate one-way functions based on expander graphs. Technical Report 00-090, Electronic Colloquium on Computational Complexity (2000)
7. Capalbo, M., Reingold, O., Vadhan, S., Wigderson, A.: Randomness conductors and constant-degree expansion beyond the degree/2 barrier. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing, pp. 659–668 (2002)
8. Ben-Sasson, E., Wigderson, A.: Short proofs are narrow — resolution made simple. Journal of ACM 48(2), 149–169 (2001)
9. Hoory, S., Linial, N., Wigderson, A.: Expander graphs and their applications. Bulletin of the American Mathematical Society 43, 439–561 (2006)

Gate Elimination for Linear Functions and New Feebly Secure Constructions

Alex Davydow[1] and Sergey I. Nikolenko[2]

[1] St. Petersburg Academic University, ul. Khlopina, 8, korp. 3, St. Petersburg, Russia
adavydow@yandex.ru
[2] Steklov Mathematical Institute, nab. r. Fontanka, 27, St. Petersburg, Russia
sergey@logic.pdmi.ras.ru

Abstract. We consider gate elimination for linear functions and show two general forms of gate elimination that yield novel corollaries. Using these corollaries, we construct a new linear feebly secure trapdoor function that has order of security $\frac{5}{4}$ which exceeds the previous record for linear constructions. We also give detailed proofs for nonconstructive circuit complexity bounds on linear functions.

Keywords: circuit complexity, gate elimination, feebly secure cryptography, feebly one-way functions.

1 Introduction

Modern cryptography has virtually no provably secure constructions. Starting from the first Diffie–Hellman key agreement protocol [5] and the first public key cryptosystem RSA [21], not a single public key cryptographic protocol has been proven secure (however, there exist secure secret key protocols, e.g., the one-time pad scheme [25, 23]). Naturally, an unconditional proof of security would be indeed hard to find, since it would necessarily imply that P $\neq$ NP. But the situation is worse: there are also no conditional proofs that might establish a connection between natural structural assumptions (like P$\neq$NP or BPP$\neq$NP) and cryptographic security. Recent developments in lattice-based cryptosystems relate cryptographic security with worst-case complexity, but they deal with problems unlikely to be NP-complete [1,6,19,20].

There are known complete cryptographic constructions, both one-way functions [14,15] and public key cryptosystems [9,8]. However, they also do not let us relate cryptographic security to key assumptions of classical complexity theory. Moreover, the asymptotic nature of these completeness results does not let us say anything about how hard it is to break a given cryptographic protocol for keys of a certain fixed length, which is, in fact, what we all want as privacy-aware customers. Problems that have been studied extensively in relation to cryptography (factoring and discrete logarithm) do seem to scale well, but there are no lower bounds and little hope for such anytime soon, so the point is moot. We can only say that complexity of the algorithms that we have devised ourselves scales well, so ultimately we fall back on the "many smart people have thought about it" argument. There are other dangers on the way of asymptotic complexity, too [13]. Ultimately, we do not care whether a protocol can or cannot be broken in the limit; we would be very happy if breaking this specific version of the protocol required constant time, but the constant was larger than the size of the known Universe.

However, modern cryptography is still very far away from *provably* secure constructions. At present, we can prove security neither in this "hard" sense nor in the sense of classical cryptographic definitions [7]. Nevertheless, while we are unable to prove a superpolynomial gap between the complexities of honest parties and adversaries, we are able to prove *some* gap. In 1992, Alain Hiltgen [10] presented a series of linear functions that are about *twice* ($1 - \epsilon$ times for an arbitrarily small ϵ) harder to invert than to compute. His example consists of linear functions over $\mathbb{F}_2$ with matrices that have few non-zero entries (ones) while inverse matrices have many ones. The complexity gap follows by a simple argument of Lamagna and Savage [16,22]: every bit of the output depends nonidly on many variables and all these bits correspond to different functions, hence a lower bound on the complexity of computing them all together. The model of computation here is the most general one, namely the number of gates in a Boolean circuit that uses arbitrary binary Boolean gates. Little more could be expected for this model at present. For example, the best known lower bound for general circuit complexity of a specific Boolean function is $3n - o(n)$ [3,26]. In his thesis, Hiltgen also presented a feebly secure function that is exactly twice harder to invert than to compute, this time a nonlinear one [11].

Lately, in the works of Hirsch, Nikolenko, and Melanich new cryptographic constructions with the same properties were constructed [12,17]. In [12], a linear feebly secure trapdoor function with order of security $\frac{25}{22}$ was constructed, and in [17], a nonlinear feebly secure trapdoor function with order of security $\frac{7}{5}$. In this paper, we continue that line of work. In Section 2, we give basic definitions.

Virtually all bounds in general circuit complexity have been proven with *gate elimination*. This paper deals with gate elimination for linear functions; in Section 3, we distill gate elimination to two basic ideas, formulate it in a general form, note important corollaries, and discuss its limitations. Our discussion in Section 3 generalizes the methods of [12], and this understanding lets us find a better construction of a linear feebly secure trapdoor function that has order of security $\frac{5}{4} - \epsilon$ for any $\epsilon > 0$, as shown in Section 4. In Section 5, we compare what we have found with nonconstructive upper and lower bounds on general circuit complexity of linear functions. Section 6 concludes the paper.

2 Preliminaries

We denote by $\mathbb{B}_{n,m}$ the set of all 2^{m2^n} total functions $f : \mathbb{B}^n \to \mathbb{B}^m$, where $\mathbb{B} = \{0, 1\}$ is the field with two elements. A *circuit* is a directed acyclic labeled graph with vertices of two kinds: vertices of indegree 0 (vertices that no edges enter) labeled by one of the variables $x_1, \ldots, x_n$, and vertices labeled by a binary Boolean function $f \in \mathbb{B}_{2,1}$; this model of computation is known as *general circuit* *complexity*. Vertices of the first kind are called *inputs* or input variables; vertices of the second kind, *gates*. The *size* of a circuit C, $\mathrm{size}(C)$, is the number of gates in it. We assume that each gate in this circuit depends of both inputs, i.e., there are no gates marked by constants and unary functions Id and $\neg$. To safely remove such gates without loss of generality, we assume that the output of each gate can be both the value of corresponding function and its negation. For every injective function of n variables $f_n \in \mathbb{B}_{n,m}$ we define its *measure of one-wayness* $M_F(f_n) = \frac{C(f_n^{-1})}{C(f_n)}$. Hiltgen's work was to find sequences of functions $f = \{f_n\}_{n=1}^{\infty}$ with a large asymptotic constant $\liminf_{n\to\infty} M_F(f_n)$, which is called the *order of one-wayness* of f.

There is a well-known definition in cryptography for a family of trapdoor functions [7]. However, we have a more detailed definition: since we are interested in constants here, we must pay attention to all the details.

Definition 1. *Fix functions* $\mathrm{pi}, \mathrm{ti}, m, c : \mathbb{N} \to \mathbb{N}$. *A* feebly trapdoor candidate *is a sequence of triples of circuits* $\mathcal{C} = \{(\mathrm{Key}_n, \mathrm{Eval}_n, \mathrm{Inv}_n)\}_{n=1}^{\infty}$ *where:*

– $\{\mathrm{Key}_n\}_{n=1}^{\infty}$ *is a family of sampling circuits* $\mathrm{Key}_n : \mathbb{B}^n \to \mathbb{B}^{\mathrm{pi}(n)} \times \mathbb{B}^{\mathrm{ti}(n)}$,
– $\{\mathrm{Eval}_n\}_{n=1}^{\infty}$ *is a family of evaluation circuits* $\mathrm{Eval}_n : \mathbb{B}^{\mathrm{pi}(n)} \times \mathbb{B}^{m(n)} \to \mathbb{B}^{c(n)}$, *and*
– $\{\mathrm{Inv}_n\}_{n=1}^{\infty}$ *is a family of inversion circuits* $\mathrm{Inv}_n : \mathbb{B}^{\mathrm{ti}(n)} \times \mathbb{B}^{c(n)} \to \mathbb{B}^{m(n)}$

such that for every security parameter n, every seed $s \in \mathbb{B}^n$, and every input $\mathrm{m} \in \mathbb{B}^{m(n)}$

$$\mathrm{Inv}_n(\mathrm{Key}_{n,2}(s), \mathrm{Eval}_n(\mathrm{Key}_{n,1}(s), \mathrm{m})) = \mathrm{m},$$

where $\mathrm{Key}_{n,1}(s)$ *and* $\mathrm{Key}_{n,2}(s)$ *are the first* $\mathrm{pi}(n)$ *bits ("public information") and the last* $\mathrm{ti}(n)$ *bits ("trapdoor information") of* $\mathrm{Key}_n(s)$, *respectively.*

Informally speaking, n is the security parameter (the length of the random seed), $m(n)$ is the length of the input to the function, $c(n)$ is the length of the function's output, and $\mathrm{pi}(n)$ and $\mathrm{ti}(n)$ are lengths of the public and trapdoor information, respectively. We call these functions "candidates" because Definition 1 does not imply any security, it merely sets up the dimensions and provides correct inversion. In our constructions, $m(n) = c(n)$ and $\mathrm{pi}(n) = \mathrm{ti}(n)$.

To find how secure a function is, we introduce the notion of a break. Informally, an adversary should invert the function without knowing the trapdoor information. We introduce break as inversion with probability greater than a certain constant r (we will usually set r to equal $\frac{1}{2}$, $\frac{3}{4}$, or $\frac{7}{8}$). We denote by $C_\alpha(f)$ the minimal size of a circuit that correctly computes a function $f \in \mathcal{B}_{n,m}$ on more than fraction α of its inputs (of length n). Obviously, $C_\alpha(f) \le C_\beta(f)$ for all $\alpha \le \beta$, and $C(f) = C_1(f)$.

Definition 2. *A circuit N* breaks *a feebly trapdoor candidate* $\mathcal{C} = \{\mathrm{Key}_n, \mathrm{Eval}_n, \mathrm{Inv}_n\}$ *on seed length n with probability r if, for uniformly chosen seeds $s \in \mathbb{B}^n$ and inputs $m \in \mathbb{B}^{m(n)}$,*

$$\Pr_{(s,m)\in U}\left[N(\mathrm{Key}_{n,1}(s), \mathrm{Eval}_n(\mathrm{Key}_{n,1}(s), m)) = m\right] > r.$$

Definition 3. *A feebly trapdoor candidate* $\mathcal{C} = \{\mathrm{Key}_n, \mathrm{Eval}_n, \mathrm{Inv}_n\}$ *has* order of security k *with level* α *if for every sequence of circuits* $\{N_n\}_{n=1}^{\infty}$ *that break* f *on every input length* n *with probability* α*,*

$$\lim \inf_{n\to\infty} \min \left\{ \frac{C(N_n)}{C(\mathrm{Key}_n)}, \frac{C(N_n)}{C(\mathrm{Eval}_n)}, \frac{C(N_n)}{C(\mathrm{Inv}_n)} \right\} \geq k.$$

In other words,

$$\lim \inf_{n\to\infty} \min \left\{ \frac{C_{3/4}(f_{\mathrm{pi}(n)+c(n)})}{C(\mathrm{Key}_n)}, \frac{C_{3/4}(f_{\mathrm{pi}(n)+c(n)})}{C(\mathrm{Eval}_n)}, \frac{C_{3/4}(f_{\mathrm{pi}(n)+c(n)})}{C(\mathrm{Inv}_n)} \right\} \geq k,$$

where the function $f_{\mathrm{pi}(n)+c(n)}$ *maps* $\left(\mathrm{Key}_{n,1}(s), \mathrm{Eval}_n(\mathrm{Key}_{n,1}(s), m)\right) \mapsto m$.

We list a few simple examples. If there is no secret key at all ($\mathrm{ti}(n) = 0$), each feebly trapdoor candidate $\{(\mathrm{Key}_n, \mathrm{Eval}_n, \mathrm{Inv}_n)\}_{n=1}^{\infty}$ has order of security 1, since the sequence of circuits $\{\mathrm{Inv}_n\}_{n=1}^{\infty}$ successfully inverts it. If a sequence $\{(\mathrm{Key}_n, \mathrm{Eval}_n, \mathrm{Inv}_n)\}_{n=1}^{\infty}$ implements a trapdoor function in the usual cryptographic sense, then $k = \infty$. Moreover, $k = \infty$ even if the bounds on adversary size are just a bit more than linear, e.g., if the adversary requires $O(n \log n)$ gates. Our definitions are not designed to distinguish between these (very different) cases, because, unfortunately, any nonlinear lower bound on general circuit complexity appears very far away from our current state of knowledge.

Let us also note explicitly that we are talking about *one-time* security. An adversary can amortize his circuit complexity on inverting a feebly trapdoor candidate for the second time for the same seed, for example, by computing the trapdoor information and successfully reusing it. Thus, in this setting one has to pick a new seed for every input.

3 Gate Elimination for Linear Functions

Gate elimination is virtually the only method we have to prove lower bounds in general circuit complexity; so far, it has been used for every single lower bound [26,3,18,24]. The basic idea of this method is to use the following inductive argument. Consider a function f and a circuit of minimal size C that computes it. Now substitute some value c for some variable x thus obtaining a circuit for the function $f\mid_{x=c}$. The original circuit C can now be simplified, because the gates that had this variable as inputs become either unary (recollect that the negation can be embedded into subsequent gates) or constant (in this case we can even proceed to eliminating subsequent gates). After figuring out how many gates one can eliminate on every step, one proceeds by induction as long as it is possible to find a suitable variable that eliminates enough gates. Evidently, the number of eliminated gates is a lower bound on the complexity of f.

Usually, the important case here is when a gate is nonlinear, such as an AND or an OR gate. In that case, it is always possible to choose a value for an input of such a gate so that this gate becomes a constant and, therefore, its immediate descendants can also be eliminated. However, in this paper we deal with gate elimination for *linear* functions. We do not know how to prove that one cannot, in general, produce a smaller circuit for a linear function with nonlinear gates, but it is evident that we cannot assume any gates to be nonlinear in this setting. Thus, gate elimination distills to two very simple ideas. Idea 1 is trivial and has been noted many times before, while Idea 2 will allow us to devise better feebly secure constructions in Section 4.

Since we are dealing with linear functions, we will, for convenience, state our results in terms of matrices over $\mathbb{F}_2$; the circuit complexity of a matrix $C_\alpha(A)$ is the circuit complexity of the corresponding linear function. By A_{-i} we denote the matrix A without its i^{th} column; note that if A corresponds to f then A_{-i} corresponds to $f\mid_{x_i=0}$. If a matrix A has a zero column A_i, it means that the corresponding function does not depend on the input x_i; in what follows, we will always assume that functions depend nontrivially on all their inputs and thus the matrices do not have zero columns; we call such matrices *nontrivial.* Note that if A is a submatrix of B then $C_\alpha(A) \le C_\alpha(B)$ for all $\alpha \in [0,1]$.

Idea 1. Suppose that for n steps, there is at least one gate to eliminate. Then $C(f) \ge n$.

Theorem 1. *Fix a real number $\alpha \in [0,1]$. Suppose that $\mathcal{P} = \{P_n\}_{n=1}^{\infty}$ is a series of predicates defined on matrices over $\mathbb{F}_2$ with the following properties:*

- *if $P_1(A)$ holds then $C_\alpha(A) \ge 1$;*
- *if $P_n(A)$ holds then $P_m(A)$ holds for every $1 \le m \le n$;*
- *if $P_n(A)$ holds then, for every index i, $P_{n-1}(A_{-i})$ holds.*

Then, for every matrix A with $\ge n+1$ columns, if $P_n(A)$ holds then $C_\alpha(A) \ge n$.

Proof. The proof is a straightforward induction on the index of P_i; the first property of $\mathcal{P}$ provides the base, and other properties takes care of the induction step. For the induction step, consider the first gate of an optimal circuit C implementing A. By the monotonicity property of $\mathcal{P}$ and the induction base, the circuit is nontrivial, so there is a first gate. Consider a variable x_i entering that gate. Note that if C computes f on fraction α of its inputs then for some c, $C\mid_{x_i=c}$ computes $f\mid_{x_i=c}$ on fraction α of its inputs. If we substitute this value into this variable, we get a circuit $C\mid_{x_i=c}$ that has at most $\text{size}(C)-1$ gates and implements A_{-i} on at least α fraction of inputs. □

The theorem is absolutely trivial; however, it has so far been the only instrument available for gate elimination in linear functions. In fact, the only instrument has been an even simpler proposition that dates back to mid-1970s.

Proposition 1 ([16,22]; [10, Theorems 3 and 4]; [12, Proposition 1]).

1. *Suppose that $f : \mathbb{B}^n \to \mathbb{B}$ depends non-idly on each of its n variables, that is, for every i there exist values $a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n \in \mathbb{B}$ such that $f(a_1, \ldots, a_{i-1}, 0,\ a_{i+1}, \ldots, a_n) \ne f(a_1, \ldots, a_{i-1}, 1, a_{i+1}, \ldots, a_n)$. Then $C(f) \ge n-1$.*

2. *Let $f = (f^{(1)}, \ldots, f^{(m)}) : \mathbb{B}^n \to \mathbb{B}^m$, where $f^{(k)}$ is the k^{th} component of f. If the m component functions $f^{(i)}$ are pairwise different and each of them satisfies $C(f^{(i)}) \geq c \geq 1$ then $C(f) \geq c + m - 1$.*

The proof is given in [12]. Note that for linear functions, statement 1 of Proposition 1 follows from Theorem 1 for $P_n(A) =$ "A has a row with $n+1$ ones". We also derive another corollary.

Corollary 1. *If A is a matrix of rank n, and each column of A has at least two ones, then $C(A) \geq n - 2$.*

Proof. Take $P_n(A) =$"rank$(A) \geq n+2$ and each column of A has at least 2 ones". □

Idea 2. Suppose that for n steps, there exists an input in the circuit with two outgoing edges, and, moreover, in m of these cases both of these edges go to a gate (rather than a gate and an output). Then $C(f) \geq n + m$.

Theorem 2. *We call a nonzero entry* unique *if it is the only nonzero entry in its row. Fix a real number $\alpha \in [0,1]$. Suppose that $\mathcal{P} = \{P_n\}_{n=1}^{\infty}$ is a series of predicates defined on matrices over $\mathbb{F}_2$ with the following properties:*

- *if $P_1(A)$ holds then $C(A) \geq 1$;*
- *if $P_n(A)$ holds then $P_m(A)$ holds for every $1 \leq m \leq n$;*
- *if $P_n(A)$ holds then, for every index i, if the i^{th} column has no unique entries then $P_{n-2}(A_{-i})$ holds, otherwise $P_{n-1}(A_{-i})$ holds.*

Then, for every matrix A with $\geq n+1$ different columns, if $P_n(A)$ holds for some n then $C(A) \geq n$ and, moreover, $C_{\frac{3}{4}}(A) \geq n$.

Proof. We argue by induction on n; for $n = 1$ the statement is obvious.

Consider the first gate g in the optimal circuit implementing A. Since g is first, its incoming edges come from the inputs of the circuit, denote them by x_i and x_j. There are three cases.

1. One of the input variables of g, say x_i, goes directly to an output y_k. Then by setting x_i to a constant we can eliminate one gate. however, in this case y_k corresponds to a row with only one nonzero element, so ith colum has a unique element, so $P_{n-1}(A_{-i})$ hold. Therefore, we invoke the induction hypothesis as $C(A_{-i}) \geq n - 1$ and get the necessary bound.

2. One of the input gate of g, say x_i, goes to another gate. Then by setting x_i to a constant we can eliminate two gates, by properties of P_n $P_{n-2}(A_{-i})$ holds, so we invoke the induction hypothesis as $C(A_{-i}) \geq n - 2$.

3. Neither x_i nor x_j enters any other gate or output. In this case, A is a function of neither x_i nor x_j but only $g(x_i, x_j)$; we show that this cannot be the case for a function computing A on more than $\frac{3}{4}$ of the inputs. A itself depends on x_i and x_j separately because all of its columns are different; in particular, for one of these variables, say x_i, there exists an output y_k that depends only on x_i: $y_k = x_i \oplus \bigoplus_{x \in X} x$, where $x_j \notin X$. On the other hand, since every gate

in an optimal circuit nontrivially depends on both inputs, there exist values a and b such that $g(0,a) = g(1,b)$. Thus, for every assignment of the remaining variables, either on input strings with $(x_i = 0, x_j = a)$ or on input strings with $(x_i = 1, x_j = b)$ the circuit makes a mistake, which makes it wrong on at least $\frac{1}{4}$ of all inputs. □

Note that Theorem 2 directly generalizes and strengthens Theorem 1.

Corollary 2. *Fix a real number $\alpha \in [0,1]$. Suppose that $\mathcal{R} = \{R_n\}_{n=1}^{\infty}$ and $\mathcal{Q} = \{Q_m\}_{m=1}^{\infty}$ are two series of predicates defined on matrices over $\mathbb{F}_2$ with the following properties:*

- *if $R_1(A)$ holds then $C(A) \geq 1$;*
- *if $R_n(A)$ holds then $R_k(A)$ holds for every $1 \leq k \leq n$;*
- *if $R_n(A)$ holds then, for every i, $R_{n-1}(A_{-i})$ holds;*
- *if $Q_1(A)$ holds then $C(A) \geq 1$;*
- *if $Q_m(A)$ holds then $Q_k(A)$ holds for every $1 \leq k \leq n$;*
- *if $Q_m(A)$ holds then, for every i, $Q_{m-1}(A_{-i})$ holds;*
- *if $Q_m(A)$ holds and A_{-i} has more zero rows than A (i.e., removing the i^{th} column has removed the last nonzero element from at least one row) then $Q_m(A_{-i})$ holds.*

Then, for every matrix A with $\geq n+1$ columns, all of whose columns are different, if $R_n(A)$ and $Q_m(A)$ hold for some $n \geq m$ then $C(A) \geq n+m$ and, moreover, $C_{\frac{3}{4}}(A) \geq n+m$.

Proof. Immediately follows from Theorem 2 for $P_n(A) = \exists k R_k(A) \wedge Q_{n-k}(A)$.

Corollary 3 ([12, Lemma 5]). *Let $t, u \geq 1$. Assume that χ is a linear function with matrix A over $\mathbb{F}_2$. Assume also that all columns of A are different, every row of A has at least u nonzero entries, and after removing any t columns of A, the matrix still has at least one row containing at least two nonzero entries. Then $C(\chi) \geq u+t$ and, moreover, $C_{3/4}(\chi) \geq u+t$.*

Proof. Take $P_n(A) =$"After removing any n columns of A, it still has at least one nonzero row", $Q_0(A) =$"true", and $Q_m(A) =$"Every row of A has at least $m+1$ ones" for $m > 0$. Then $P_{t+1}(A)$ and $Q_{u-1}(A)$ hold, and $\mathcal{P}$ and $\mathcal{Q}$ satisfy the conditions of Corollary 2, which gives the desired bound. Note that in this case, Q_m for $m > 0$ cannot hold for a matrix where a row has only a single one, so in the gate elimination proof, for the first $u-1$ steps two gates will be eliminated, and then for $t-u+2$ steps, one gate will be eliminated. □

We also derive another, even stronger corollary that will be important for new feebly secure constructions.

Corollary 4. *Let $t \geq u \geq 2$. Assume that A is a $u \times t$ matrix with different columns, and each column of A has at least two nonzero elements (ones). Then $C(A) \geq 2t-u$ and, moreover, $C_{\frac{3}{4}}(A) \geq 2t-u$.*

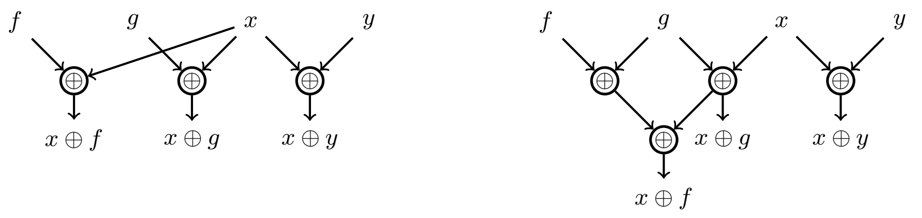


Fig. 1. Rewiring linear circuits

Proof. Take $P_n(A) =$"twice the number of nonzero columns in A less the number of nonzero rows in A is at least n". Then $P_{2t-u}(A)$ holds, and $\mathcal{P}$ satisfy the conditions of Theorem 2. $\square$

Naturally, we could prove Corollaries 1 and 4 directly. We have chosen the path of generalization for two reasons: one, to make Theorem 3 more precise and more general, and two, to show the limits of gate elimination for linear functions. As we have already mentioned, for linear functions we cannot count on nonlinear gates that could eliminate their descendants. In Theorems 1 and 2, we have considered two basic cases: when there is only one edge outgoing from a variable and when there are two edges (going either to two gates or to a gate and an output).

Figure 1 shows why we cannot expect anything more, e.g., a variable with three outgoing edges. On the left, Figure 1 shows a part of a circuit where a variable x has three outgoing edges. On the right, a rewiring of the same circuit that has all the same outputs but x only enters two gates. Naturally, this does not prove anything: we have introduced a new gate (so the circuit is no longer optimal) and have only relocated the extra input from x to an extra input from f. However, this simple example does show that to get better bounds, simple local gate elimination does not suffice, and one has to consider global properties of the function and the corresponding optimal circuit.

We finish this section with an extension of these results to block diagonal matrices. In general, we cannot prove that the direct sum of several functions has circuit complexity equal to the sum of the circuit complexities of these functions; counterexamples are known as "mass production" [26]. However, for linear functions and gate elimination in the flavours of Theorems 1 and 2, we can. The following theorem generalizes Lemma 6 of [12].

Theorem 3. *Suppose that a linear function χ is given by a block diagonal matrix*

$$\begin{pmatrix} A_1 & 0 & \cdots & 0 \\ 0 & A_2 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & A_k \end{pmatrix},$$

and every A_j satisfies the conditions of Theorem 2 with predicates $\mathcal{P}^j = \{P_n^j\}_{n=1}^{\infty}$, and $P_{n_j}^j(A_j)$ hold for every j. Then $C(\chi) \geq \sum_{j=1}^{k} n_j$.

Proof. We invoke Theorem 2 with the predicate composed of original predicates:

$$P_n = \bigvee_{i_1+\ldots+i_k=n} P^1_{i_1} \wedge P^2_{i_2} \wedge \ldots \wedge P^k_{i_k}.$$

It is now straightforward to check that $\mathcal{P} = \{P_n\}_{n=1}^{\infty}$ satisfies the conditions of Theorem 2 (since every deleted column affects only one block), and the block diagonal matrix satisfies $P_{n_1+\ldots+n_k}$. □

4 A New Linear Feebly Secure Trapdoor Function

In our constructions, we follow the general idea of [12]: first, we find a feebly trapdoor candidate that has the adversary work harder than function inversion but function evaluation is even harder. Then, we add a feebly secure one-way function as a separate block and thus reduce the work needed for function evaluation; this construction has been discussed in detail in [12].

We begin with some preliminaries. By U_n, we denote the upper triangular square $n \times n$ matrix with a bidiagonal inverse:

$$U_n = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ 0 & 1 & \cdots & 1 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 \end{pmatrix}, \qquad U_n^{-1} = \begin{pmatrix} 1 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix};$$

note that U_n^2 is an upper triangular matrix with zeros and ones chequered. In what follows, we often write matrices that consist of other matrices as blocks; e.g., $\begin{pmatrix} U_n & U_n \end{pmatrix}$ is an $n \times 2n$ matrix consisting of two upper triangular blocks.

Lemma 1. *1.* $C_{\frac{3}{4}}(U_n) = n - 1$.
2. $C_{\frac{3}{4}}(U_n^2) = n - 2$.
3. $C_{\frac{3}{4}}(U_n^{-1}) = n - 1$.
4. $C_{\frac{3}{4}}(\begin{pmatrix} U_n & U_n \end{pmatrix}) = 2n - 1$.
5. $3n - 6 \le C_{\frac{3}{4}}(\begin{pmatrix} U_n^2 & U_n \end{pmatrix}) \le C(\begin{pmatrix} U_n^2 & U_n \end{pmatrix}) \le 3n - 3$.
6. $3n - 4 \le C_{\frac{3}{4}}(\begin{pmatrix} U_n & U_n^{-1} \end{pmatrix}) \le C(\begin{pmatrix} U_n & U_n^{-1} \end{pmatrix}) \le 3n - 2$.

Proof. Lower bounds in items 1–3 are trivial: every row is different and no inputs except one (two for 2) are connected to outputs directly. Thus, we need at least one gate per row. The lower bound for item 4 follows from simple counting: the first row of this matrix has $2n$ nonzero entries, so at least $2n - 1$ gates are needed to compute it. The lower bound for item 5 (respectively, 6) follows by Corollary 4: the matrix $\begin{pmatrix} U_n^2 & U_n \end{pmatrix}$ (resp., $\begin{pmatrix} U_n & U_n^{-1} \end{pmatrix}$) satisfies its assumptions except for three (resp., two) columns, so the corollary is invoked for $t = 2n - 3$ (resp., $t = 2n - 2$) and $u = n$.

We prove upper bounds by providing explicit circuit constructions. To compute 1, note that every row differs from the previous one only in a single position, so we can compute each output out_i as $out_{i+1} \oplus in_i$. Moreover, $out_n = in_n$ so we need no gates for it. The same idea applies in 2, but in this case out_n and

out_{n-1} are computed directly, and $out_i = out_{i-2} \oplus in_i$. To compute 3, we simply compute each row independently. To compute 4, we apply an idea from [12]. Note that $\begin{pmatrix} U_n & U_n \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = U_n \cdot a \oplus U_n \cdot b = U_n \cdot (a \oplus b)$. We use n gates to compute $a \oplus b$ and then compute the result using $n-1$ gates. To compute 5 and 6, note that $\begin{pmatrix} A & B \end{pmatrix} \cdot \begin{pmatrix} a \\ b \end{pmatrix} = A \cdot a \oplus B \cdot b$. Thus, we can divide each of these computations in two parts which can be computed independently using previous algorithms, and then use n gates to compute the final XOR. □

For the first construction, we assume that lengths of public information pi, trapdoor information ti, message m, and the cipher c are the same and equal n. We let $ti = U_n \cdot pi$, $c = \begin{pmatrix} U_n^{-1} & U_n \end{pmatrix} \cdot \begin{pmatrix} m \\ pi \end{pmatrix}$. In this case, an adversary would have to compute the matrix $\begin{pmatrix} U_n & U_n \end{pmatrix} \cdot \begin{pmatrix} c \\ ti \end{pmatrix} = \begin{pmatrix} U_n & U_n^2 \end{pmatrix} \cdot \begin{pmatrix} c \\ pi \end{pmatrix}$. Now, inversion without the trapdoor is harder than inversion with trapdoor, but encryption is about the same complexity as inversion without trapdoor, so we cannot call it a feebly trapdoor function yet.

To solve this problem, we consider a feebly one-way linear function A and construct the protocol in the following way (I_n is the identity matrix here):

$$\begin{aligned}
\mathrm{Key}_n &= \begin{pmatrix} U_n & 0 \\ 0 & I_n \end{pmatrix} \cdot \begin{pmatrix} s & s \end{pmatrix} = \begin{pmatrix} t_i \\ p_i \end{pmatrix}, \\
\mathrm{Eval}_n &= \begin{pmatrix} U_n^{-1} & U_n & 0 \\ 0 & 0 & A \end{pmatrix} \cdot \begin{pmatrix} m_1 \\ pi \\ m_2 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}, \\
\mathrm{Inv}_n &= \begin{pmatrix} U_n & U_n & 0 \\ 0 & 0 & A^{-1} \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ ti \\ c_2 \end{pmatrix} = \begin{pmatrix} m_1 \\ m_2 \end{pmatrix}.
\end{aligned}$$

The adversary's problem now becomes to compute

$$\mathrm{Adv}_n = \begin{pmatrix} U_n & U_n^2 & 0 \\ 0 & 0 & A^{-1} \end{pmatrix} \cdot \begin{pmatrix} c_1 \\ pi \\ c_2 \end{pmatrix} = \begin{pmatrix} m_1 \\ m_2 \end{pmatrix}.$$

For the feebly one-way function A, we fix a small $\epsilon > 0$ and take the Hiltgen's linear function with order of security $2-\epsilon$ [10]; we take its size to be λn with λ chosen below. In Hiltgen's constructions, it means that $C_{\frac{3}{4}}(A) = \lambda n + o(n)$, and $C_{\frac{3}{4}}(A^{-1}) = (2-\epsilon)\lambda n + o(n)$. Now Lemma 1 and Theorem 3 imply the following complexity bounds:

$$\begin{aligned}
C_{\frac{3}{4}}(\mathrm{Key}_n) &= n-1, \\
C_{\frac{3}{4}}(\mathrm{Eval}_n) &= 3n + \lambda n + o(n) &&= (3+\lambda)n + o(n), \\
C_{\frac{3}{4}}(\mathrm{Inv}_n) &= 2n + (2-\epsilon)\lambda n + o(n) &&= (2 + (2-\epsilon)\lambda)n + o(n), \\
C_{\frac{3}{4}}(\mathrm{Adv}_n) &= 3n + (2-\epsilon)\lambda n + o(n) &&= (3 + (2-\epsilon)\lambda)n + o(n).
\end{aligned}$$

The order of security of this construction is now

$$\lim_{n\to\infty}\left(\min\left(\frac{C_{3/4}(\mathrm{Adv}_n)}{C(\mathrm{Eval}_n)}, \frac{C_{3/4}(\mathrm{Adv}_n)}{C(\mathrm{Inv}_n)}, \frac{C_{3/4}(\mathrm{Adv}_n)}{C(\mathrm{Key}_n)}\right)\right) = \\ = \min\left(\frac{3+(2-\epsilon)\lambda}{3+\lambda}, \frac{3+(2-\epsilon)\lambda}{2+(2-\epsilon)\lambda}\right).$$

This expression reaches maximum for $\lambda = \frac{1}{1-\epsilon}$, and this maximum is $\frac{5-4\epsilon}{4-\epsilon}$, which tends to $\frac{5}{4}$ as $\epsilon \to 0$. Thus, we have proven the following theorem.

Theorem 4. *For every $\epsilon > 0$, there exists a linear feebly trapdoor function with seed length $\mathrm{pi}(n) = \mathrm{ti}(n) = n$, length of inputs and outputs $c(n) = m(n) = 2n$, and order of security $\frac{5}{4} - \epsilon$.*

In Theorem 3, we have generalized a hardness amplification procedure similar to [12, Theorem 2]; with it, we can obtain superpolynomial security guarantees against weaker adversaries.

Theorem 5. *For every $\epsilon > 0$, there exists a linear feebly trapdoor function with seed length $\mathrm{pi}(n) = \mathrm{ti}(n) = n$, length of inputs and outputs $c(n) = m(n) = 2n$, complexities $C_{\frac{3}{4}}(\mathrm{Key}_n) = n-1$, $C_{\frac{3}{4}}(\mathrm{Eval}_n) = 4n+o(n)$, and $C_{\frac{3}{4}}(\mathrm{Inv}_n) = \frac{4-\epsilon}{1-\epsilon}n + o(n)$, and order of security $\frac{5-4\epsilon}{4-\epsilon}$. Moreover, no adversary with less than $\frac{5-4\epsilon}{1-\epsilon}n - \frac{5}{2}\delta\sqrt{n}$ gates can invert this feebly trapdoor function on more than $2^{-\delta\sqrt{n}+o(\sqrt{n})}$ of its inputs for any constant $\delta > 0$.*

Proof. We consider the block diagonal matrix

$$H = \begin{pmatrix} X & 0 & \dots & 0 \\ 0 & X & \dots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \dots & X \end{pmatrix},$$

with m diagonal blocks, where X is the matrix of the trapdoor function constructed in Theorem 4, and apply Theorem 3. Stacking the matrices up in a large block diagonal matrix does not change the parameters of a feebly trapdoor function. □

5 Nonconstructive Bounds for Linear Functions

In Section 3, we have seen that we cannot currently hope to prove more than linear lower bounds on general circuit complexity; the same is true, of course, for linear functions. A classical result shows by counting that among general Boolean functions of n variables, almost all of them have circuit complexity $\geq \frac{1}{n}2^n$. But maybe for the linear case, nonlinear bounds are impossible from the beginning?

It turns out that linear functions with nonlinear bounds do exist. References to this result can be found [2,4], but we have not been able to find a detailed proof in literature, so we include it here and refine it to get exact constants.

Theorem 6. *1. For every n there exists a constant δ_n such that the circuit complexity of all linear functions $\phi : \{0,1\}^n \to \{0,1\}^n$ does not exceed $\delta_n \frac{n^2}{\log n}$, and $\lim_{n\to\infty} \delta_n = 1$.*

2. For every $n \geq 3$, there exists a linear Boolean function $\phi : \{0,1\}^n \to \{0,1\}^n$ with circuit complexity greater than $\frac{n^2}{2\log n}$.

Proof. 1. *Upper bound.* Let A be the matrix of ϕ. For clarity, we assume n is a power of 2 (the same proof goes through with very minor modifications if n is not). We implement A as follows. First, we generate c_i as all possible rows of zeros and ones of length $l = q \log n$, where q is a constant to be selected later. Denoting the inputs of ϕ by $x = \left(x_1 \cdots x_n \right)$, we preprocess the values of all possible combinations of $c_i \cdot \begin{pmatrix} x_{j+1} \\ \cdots \\ x_{j+l} \end{pmatrix}$, where j is a multiple of l. The total number of gates needed for this operation is bounded from above by

$$2^{q \cdot \log n} \cdot \frac{n}{q \cdot \log n} \cdot (q \log n - 1) \le n^q \cdot n = n^{q+1}.$$

Let $A_1, A_2, \cdots, A_n$ be the columns of A. To find $A \cdot x$, we compute

$$A \cdot x = \left(A_1 \dots A_l \right) \begin{pmatrix} x_1 \\ \vdots \\ x_l \end{pmatrix} \oplus \left(A_{l+1} \cdots A_{2l} \right) \begin{pmatrix} x_{l+1} \\ \vdots \\ x_{2l} \end{pmatrix} \oplus \dots \oplus \left(A_{n-l+1} \cdots A_n \right) \begin{pmatrix} x_{n-l+1} \\ \vdots \\ x_n \end{pmatrix}.$$

After preprocessing, every product in this formula will already be computed, and all we need to do is to choose a correct wire, so no gates are needed here. After that, $n \cdot (\frac{n}{l} - 1)$ gates are needed to XOR for the total result. Thus, the total number of gates needed is

$$n^{q+1} + n \cdot (\frac{n}{q \cdot \log n} - 1) = n^{q+1} + \frac{n^2}{q \cdot \log n} - n.$$

Setting $q = \frac{1}{1+\epsilon}$, we get the upper bound $(1+\epsilon)\frac{n^2}{\log n}$ for an arbitrarily small ϵ.

2. *Lower bound.* The lower bound is proven by counting. Let $q = (1-\epsilon)\frac{n^2}{2 \log n}$. We estimate T, the number of circuits of size $\le q$. There are 16 types of gates, and every circuit's description consists of type and inputs for every gate. Therefore,

$$T \le \frac{q \cdot (16 \cdot (n+q)^2))^q}{q!} \le q \cdot (16e)^q \cdot \frac{(n+q)^{2q}}{q^q} \le q \cdot (64e)^q \cdot \frac{q^{2q}}{q^q} = (64 \cdot e \cdot q)^q \le$$
$$\le (64 \cdot e \cdot q)^{q+1} \le (n^2)^{(1-\epsilon) \cdot \frac{n^2}{2 \log n} + 1} = (2^{2 \log n})^{(1-\epsilon) \cdot \frac{n^2}{2 \log n} + 1} = 2^{(1-\epsilon)n^2 + 2 \log n}.$$

But the total number of linear Boolean functions of n arguments is 2^{n^2}, which is greater than $2^{(1-\epsilon)n^2 + 2 \log n}$. Therefore, there are Boolean functions with circuit complexity exceeding $\frac{n^2}{2 \log n}$. □

6 Conclusion

In this paper, we have discussed in detail the circuit complexity of linear Boolean functions. We have proven two general statements on gate elimination in linear functions, derived several corollaries, and applied them to find a new linear feebly secure trapdoor function, with better order of security than the known one [12]. While feebly secure cryptographic primitives can hardly be put to any practical use, they are still important from the theoretical point of view. As sad as it sounds, this is actually the frontier of provable, mathematically sound results on security; we do not know how to prove anything stronger. However, in Section 5 we have seen that with these bounds, we are only scratching the surface even for linear Boolean functions, let alone nonlinear ones.

Further work in this direction is twofold. One can further develop the notions of feebly secure primitives. Orders of security can certainly be improved; perhaps, new primitives (key agreement protocols, zero knowledge proofs etc.) can find their feebly secure counterparts. This work can widen the scope of feebly secure methods, but the real breakthrough can only come from one place.

It becomes clear that cryptographic needs call for further advances in general circuit complexity. General circuit complexity has not had a breakthrough since the 1980s; nonconstructive lower bounds are easy to prove by counting, but constructive lower bounds remain elusive. The best bound we know is $3n - o(n)$, proven in 1984 [3]. At present, we do not know how to rise to this challenge; none of the known methods seem to work, so a general breakthrough is required for nonlinear lower bounds on circuit complexity. The importance of such a breakthrough can hardly be overstated; feebly secure cryptographic constructions provide yet another application for new circuit lower bounds.

Acknowledgements

We thank Olga Melanich and the anonymous referees for pointing out important shortcomings in a preliminary version of our paper.

Work of the first author has been supported by the Yandex Stipend Program. Work of the second author has been supported by the Russian Presidential Grant Programme for Young Ph.D.'s, grant no. MK-4089.2010.1, for Leading Scientific Schools, grant no. NSh-5282.2010.1, Federal Target Programme "Scientific and scientific-pedagogical personnel of the innovative Russia", and Russian Fund for Basic Research grants.

References

1. Ajtai, M., Dwork, C.: A public-key cryptosystem with worst-case/average-case equivalence. In: Proceedings of the 29th Annual ACM Symposium on Theory of Computing, pp. 284–293 (1997)
2. Alon, N., Karchmer, M., Wigderson, A.: Linear circuits over GF(2). SIAM Journal of Computing 19(6), 1064–1067 (1990)
3. Blum, N.: A boolean function requiring $3n$ network size. Theoretical Computer Science 28, 337–345 (1984)
4. Bublitz, S.: Decomposition of graphs and monotone formula size of homogeneous functions. Acta Informatica 23, 410–417 (1986)
5. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Transactions on Information Theory IT-22, 644–654 (1976)
6. Dwork, C.: Positive applications of lattices to cryptography. In: Privara, I., Ružička, P. (eds.) MFCS 1997. LNCS, vol. 1295, pp. 44–51. Springer, Heidelberg (1997)
7. Goldreich, O.: Foundations of Cryptography. Basic Tools. Cambridge University Press, Cambridge (2001)
8. Grigoriev, D., Hirsch, E.A., Pervyshev, K.: A complete public-key cryptosystem. Groups, Complexity, and Cryptology 1, 1–12 (2009)

9. Harnik, D., Kilian, J., Naor, M., Reingold, O., Rosen, A.: On robust combiners for oblivious transfer and other primitives. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 96–113. Springer, Heidelberg (2005)
10. Hiltgen, A.P.: Constructions of feebly-one-way families of permutations. In: Proc. of AsiaCrypt 1992, pp. 422–434 (1992)
11. Hiltgen, A.P.: Cryptographically Relevant Contributions to Combinational Complexity Theory, ETH Series in Information Processing, vol. 3. Konstanz: Hartung-Gorre (1994)
12. Hirsch, E.A., Nikolenko, S.I.: A feebly secure trapdoor function. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) CSR 2009. LNCS, vol. 5675, pp. 129–142. Springer, Heidelberg (2009)
13. Koblitz, N., Menezes, A.: Another look at "provable security". Journal of Cryptology 20(1), 3–37 (2007)
14. Kojevnikov, A.A., Nikolenko, S.I.: New combinatorial complete one-way functions. In: Proceedings of the 25th Symposium on Theoretical Aspects of Computer Science, Bordeaux, France, pp. 457–466 (2008)
15. Kojevnikov, A.A., Nikolenko, S.I.: On complete one-way functions. Problems of Information Transmission 45(2), 108–189 (2009)
16. Lamagna, E.A., Savage, J.E.: On the logical complexity of symmetric switching functions in monotone and complete bases. Tech. rep., Brown University, Rhode Island (July 1973)
17. Melanich, O.: Nonlinear feebly secure cryptographic primitives. PDMI preprints 12 (2009)
18. Paul, W.J.: A $2.5n$ lower bound on the combinational complexity of boolean functions. SIAM Journal of Computing 6, 427–443 (1977)
19. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, pp. 84–93 (2005)
20. Regev, O.: Lattice-based cryptography. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 131–141. Springer, Heidelberg (2006)
21. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
22. Savage, J.E.: The Complexity of Computing. Wiley, New York (1976)
23. Shannon, C.E.: Communication theory of secrecy systems. Bell System Technical Journal 28(4), 656–717 (1949)
24. Stockmeyer, L.: On the combinational complexity of certain symmetric Boolean functions. Mathematical Systems Theory 10, 323–326 (1977)
25. Vernam, G.S.: Cipher printing telegraph systems for secret wire and radio telegraphic communications. Journal of the IEEE 55, 109–115 (1926)
26. Wegener, I.: The Complexity of Boolean Functions. B. G. Teubner, and John Wiley & Sons (1987)

Finite Groups and Complexity Theory: From Leningrad to Saint Petersburg via Las Vegas

László Babai

University of Chicago, Chicago IL 60637, USA
laci@cs.uchicago.edu
http://people.cs.uchicago.edu/∼laci

Abstract. Finite groups have affected complexity theory and complexity theory has had an impact on computational group theory. This paper is a personal account of the author's journey through the evolution of some of these interconnections, culminating in recent definitive results on the matrix group membership problem.

1 Introduction

Many examples illustrate the influence of group theory on complexity theory. It is not easy to pinpoint what it is about groups that makes them relevant, but perhaps two factors could be mentioned, one trivial, and one profound: a subgroup is at most half the size of the group, and groups tend to expand. The first, trivial fact, to which I will refer as "Lagrange's Theorem," is often the key to randomized algorithms involving groups, such as the Solovay-Strassen and Miller-Rabin primality tests. Expansion comes in two varieties: some groups show explosive expansion; and all groups expand at a non-negligible rate. Foremost among the former are the projective linear groups $\mathrm{PSL}(2, p)$, responsible for explicit graphs of large girth [66], the Ramanujan graphs [67,61], Gowers-quasirandomness in groups [50] (cf. [25]), Helfgott growth [51] (cf. [72,37]), almost certain expanders [34]. Zig-zag products, related to semidirect products of groups [4], have been introduced in an attempt to replace deep group theory with elementary combinatorics and linear algebra in the construction of expanders [74] and have led to $\mathrm{SL} = \mathrm{L}$, Omer Reingold's celebrated result in complexity theory [73].

The more modest but ubiquitous "local expansion" (see Section 4.4) found in connection with interactive proofs [12] (cf. [11,28]) plays a key role in the analysis of algorithms for groups [18,11].

Expanders have served as important derandomization tools. At the same time, groups appear to attract randomization.

The first attempt at a group theoretic approach to graph isomorphism yielded a Las Vegas algorithm [8] (and the term). The attempt to classify the complexity of matrix group membership was one of the roots of the concept of interactive proofs [9]. Randomization seems indispensable for matrix group algorithms [16].

Not all applications of groups to complexity theory fit in this framework. David Barrington famously noticed the universality of groups as computational devices: Boolean circuits can be simulated by the product and commutator operations in nonsolvable groups, resulting in a surprising simulation of NC^1 by *width-5 branching programs* of polynomial size [30].

The seminal paper by Kahn, Saks, and Sturtevant linked the *evasiveness conjecture* in decision tree complexity to group actions on simplicial complexes [56]. (Cf. [14] for recent use of a greater variety of permutation groups in this context.)

Most recently, Aaronson and his coauthors connected the group membership problem to separation of quantum complexity classes [1] and quantum vs. classical communication complexity [2].

This is but a small sample of the cases where groups played a role in complexity theory.

Conversely, a complexity theoretic approach to problems of computational group theory has profoundly affected the latter area. Largely due to the work of Ákos Seress, an increasing portion of the computational group theory package GAP [47] is now based on algorithms with rigorous performance guarantees, a hallmark of the complexity theoretic approach.

In this paper we review some of these connections through the author's personal lense. The main themes are the Graph Isomorphism problem, the membership problem in matrix groups, and randomization in complexity theory.

We assume that the reader is familiar with the basics of group theory as described, for instance, in [75].

2 Recognition vs. Construction vs. Sampling

The two most common explicit representations of finite groups are *permutation groups* and *linear groups* over finite fields. In both cases, our group G is a subgroup of a natural *ambient group* U; in the case that G is a permutation group acting on a domain Ω, $U = \mathrm{Sym}(\Omega)$ is the symmetric group (group of all permutations of Ω); in the case when G is a linear group acting on the (finite) vector space V, we have $U = \mathrm{GL}(V)$. (If $V = \mathbb{F}_q^d$ where $\mathbb{F}_q$ is the finite field of order q then $\mathrm{GL}(V)$ can be identified with the group $\mathrm{GL}(d, q)$ of nonsingular $d \times d$ matrices over $\mathbb{F}_q$.)

More generally, we shall consider subgroups of an arbitrary finite ambient group U; we assume that group operations can be performed efficiently in U. (This concept will be defined more precisely in Section 4.1.)

We say that a subgroup $G \le U$ is *known* or *given* if we have a list of generators for G; *constructing* or *finding* G means efficiently constructing a list of generators.

We say that $G \le U$ is *samplable* if we can efficiently sample elements of G from the uniform distribution; and G is *approximately samplable* if elements of G from a nearly uniform distribution can be sampled efficiently, with an arbitrarily small parameter of proximity to uniform (in any reasonable metric on distributions).

We say that a subgroup $G \le U$ is *recognizable* if we have an efficient procedure to test membership in G, i. e., to decide, for each $\pi \in U$, whether or not $\pi \in G$.

If $H \leq G$ is a subgroup, we say that H is *recognizable in* G if given G, for every element $\pi \in G$ we can decide whether or not $\pi \in H$.

What we mean by *efficient* computation will depend on the context and will need to be clarified in each case. It will usually mean (deterministic or randomized) polynomial time, possibly with access to certain oracles.

When using terms like "we can decide" or "we can compute," we shall always mean to perform these tasks efficiently. We are dealing with explicit finite objects, so decidability or computability in the classical sense will not be at issue.

It is natural to ask how the three concepts of constructibility, samplability, and recognizability relate to each other.

It is easy to show that if we can (approximately) sample a group G of order N then we can also construct G: with high probability, a set of $O(\log N)$ random elements will generate G.

Conversely, a known group G can always be approximately sampled; this fact is a central ingredient of polynomial-time algorithms in matrix groups.

Theorem 1 ([11]). *A known group G can be approximately sampled in time* $\mathrm{polylog}(|G|)$.

(This result holds in the generality of black-box groups, see Section 4.1.)

Given the equivalence of finding and approximately sampling a group, the following natural questions remain:

(a) if H is a known subgroup of G, is H recognizable?
(b) if H is a recognizable subgroup of G, can we construct H?

Question (a) is the problem of testing membership in a known subgroup. One of the continuing success stories of the complexity theory of finite groups, and one of the main themes of this writing, has been a positive answer to (a) in a growing number of contexts (see Sections 3.4, 3.6, and Theorem 7). While the answer to (b) is negative under a natural interpretation in most contexts, there are many interesting classes of recognizable subgroups where constructibility has either been affirmed or has been a central object of study.

The foremost example of a class of recognizable subgroups whose constructibility has unknown complexity status is the *automorphism groups of graphs.* The automorphism group $\mathrm{Aut}(X) \leq \mathrm{Sym}(V)$ of the graph $X = (V, E)$ is obviously a recognizable subgroup of $\mathrm{Sym}(V)$ (V is the set of vertices). The problem of constructing $\mathrm{Aut}(X)$ is equivalent, under Cook-reductions (polynomial-time Turing reductions), to the Graph Isomorphism problem [7,68].

Important examples of recognizable subgroups of a known group G include $Z(G)$, the *center* of G (the set of elements that commute with all elements of G) and $\mathrm{Rad}(G)$, the *solvable radical* of G (the largest solvable normal subgroup). Progress on constructing the radical [71,16] has been the key to the recent definitive result on the membership problem in linear groups in odd characteristic [16] (Theorem 7); and our inability to construct the radical remains the key obstacle to solving the same problem in characteristic 2.

The fundamental problem of the interaction between the concepts of recognizability and constructibility of subgroups appears first to have been highlighted in the author's 1979 Montreal tech report [8], where also the term "recognizable subgroup" seems to have made its first appearance, as a tool to solving a case of the Graph Isomorphism problem.

Certain abelian groups of number theoretic origin played a role in the design of efficient algorithms from the early days of polynomial-time theory. Indeed, the analysis of the Solovay-Strassen and Miller-Rabin randomized primality tests rests on "Lagrange's Theorem."

As far as I know, the first occurrence of general (not necessarily abelian) finite groups in the design of polynomial-time algorithms was in [8]. This paper introduced group theoretic methods to the study of Graph Isomorphism and also initiated the complexity theory of computation in finite groups. As an aside, it also introduced the term "Las Vegas algorithm" and gave an example of such an algorithm in computational group theory.

In view of the role that this tech report played in subsequent developments, we briefly review its main algorithm.

3 The Graph Isomorphism Problem in 1979: A Las Vegas Algorithm

The special case of the Graph Isomorphism problem considered in [8] concerned *vertex-colored graphs* where each color has *bounded multiplicity;* isomorphisms preserve color by definition. (The coloring need not be legal in the sense of chromatic graph theory, i. e., there is no restirction on the colors of neighbors.) Deciding isomorphism of such graphs is easily reduced to the question of finding generators for the automorphism group of a vertex-colored graph with bounded multiplicity of colors. (The multiplicity doubles under this reduction.)

3.1 Recognition vs. Sampling

Let $X = (V, E)$ be the given graph and $V = C_1 \dot\cup \dots \dot\cup C_k$ the partition of the vertices into color classes. The automorphim group $\mathrm{Aut}(X)$ is a recognizable subgroup of the ambient group $U = \mathrm{Sym}(C_1) \times \dots \times \mathrm{Sym}(C_k)$. This group is well understood; what mattered for [8] was that U could be (uniformly) sampled. The question was, can we find $\mathrm{Aut}(X)$. For this purpose, samplability of $\mathrm{Aut}(X)$ would suffice.

3.2 Tower of Groups

A key insight was that $\mathrm{Aut}(X)$ is a member of an entire chain $U = G_0 \geq G_1 \geq \dots \geq G_m = \{1\}$ of recognizable subgroups of U, where each index $|G_i : G_{i-1}|$ is small. Under these conditions, all members of the chain can be sampled efficiently (polynomial in m and linear in $\max_i |G_i : G_{i-1}|$), as described below. I called this procedure the "tower of groups method."

The algorithm proceeds by filling up tables T_i of (left) coset representatives of G_{i-1} in G_i ($i = 1, \dots, m$). Initially each T_i consists of the identity alone; once all the T_i are complete, $G = T_1T_2 \dots T_m$ and the representation of each $\rho \in G$ as $\rho = \tau_1\tau_2 \dots \tau_m$ ($\tau_i \in T_i$) is unique.

The basic subroutine of the algorithm was an operation called "sifting." (The term was coined in [45].) Sifting an element $\rho \in G$ means to try to represent it as a product $\rho = \tau_1\tau_2 \dots \tau_m$ ($\tau_i \in T_i$). There is a unique way of doing this when the tables are complete; when we discover incompleteness, we augment the table.

```
procedure sift($\rho; T_1, \dots, T_m$)
  for $i = 1$ to $m$ if $(\exists\tau_i \in T_i)(\tau_i^{-1}\rho \in G_i)$ then $\rho := \tau_i^{-1}\rho$
    else add $\rho$ to $T_i$, exit.
```

The question then is, what elements to sift. The answer given in [8] was: sample U. In this section, "random" will always refer to the uniform distribution. As a random $\rho \in U$ reaches G_{i-1} in the **for** loop, clearly it will belong to a uniform random left coset of G_i in G_{i-1}. So, with high probability, the tables will rapidly fill up and once they did, each additional random $\rho \in U$ produces random members of each G_i.

This would give a Monte Carlo algorithm. But more can be done; we can deterministically verify whether or not we are done: we only need to check the condition $|U| = \prod_{i=1^m} |T_i|$. (In our context, the order of U is known: $|U| = \prod(|C_i|!)$. This way the algorithm can never return a wrong answer, but with an arbitrarily small probability it will honestly report failure. I coined the term *Las Vegas algorithm* for this situation and pointed out that the Berlekamp - Rabin algorithm for factoring polynomials over finite fields also belonged to the Las Vegas variety.

To apply this algorithm to finding the automorphism group of a vertex-colored graph, we need to define the relevant chain of subgroups.

Let $X = (V, E)$ be the given graph and $V = C_1\dot{\cup} \dots \dot{\cup}C_k$ the partition into color classes; assume $|C_i| \le b$ for each i. Let E_{ij} denote the set of edges induced by $C_i \cup C_j$. Let us arrange the pairs $\{i, j\}$ $(1 \le i < j \le k)$ in some linear order; set $F_t = E_{ij}$ if the pair $\{i, j\}$ is the t-th pair in this linear order ($t = 1, \dots, k(k-1)/2$). Let $K_i = \bigcup_{j \le i} F_i$ $(0 \le i \le k(k-1)/2)$, and let $G_i = \mathrm{Aut}(V, K_i)$. We have $U = G_0 \ge G_1 \ge \cdots \ge G_{k(k-1)/2} = \mathrm{Aut}(X)$. It is easy to verify that $(\forall i)(|G_{i-1} : G_i| \le (b!)^2$. Let us then define $G_{k(k-1)/2+\ell}$ as the pointwise stabilizer of the first ℓ vertices of X in $\mathrm{Aut}(X)$ $(\ell = 1, \dots, |V|)$. Setting $m = k(k-1)/2 + |V|$, our subgroup chain is complete. In the tail of the series (below $G_{k(k-1)/2}$) each index is $\le b$.

3.3 Applications

The "tower of groups" method, combined with linear algebra techniques, yielded a polynomial-time algorithm to test isomorphism of graphs with bounded multiplicity of eigenvalues [21]. Although this result was only published three years later, it was in fact the original motivation for the "tower of groups" method and was announced in [8], see Section 5.

Combined with combinatorial partitioning/refinement techniques, the "tower of groups" method also produced the first moderately exponential algorithms for isomorphism of bounded degree graphs ($\exp(\widetilde{O}(\sqrt{n}))$) [8], soon to be superseded by Luks's polynomial-time algorithm [62] that uses group theory to far greater depth.

Admittedly, the "tower of groups" method uses group theory only on the most elementary level. Yet it is capable of beating very powerful "purely combinatorial" methods. Graph isomorphism heuristics often operate with partitioning/refinement methods; we can partition not only vertices ("naive refinement") or pairs of vertices (the "Weisfeiler-Leman method") but also d-tuples for any d; I called this the "d-dimensional Weisfeiler-Leman method." The cost of the method is $n^{\Theta(d)}$ where n is the number of vertices. In a remarkable 1990 development, Cai, Fürer, and Immerman [40] constructed pairs of nonisomorphic graphs that cannot be distinguished by the d-dimensional W-L method unless $d = \Omega(n)$. On the other hand, the CFI graphs can be viewed as vertex-colored graphs with color-multiplicity 4, efficiently handled by the "tower of groups" method.

3.4 "Tower of Groups" Derandomized

Soon after I publicized the "tower of groups" algorithm at STOC'79 (cf. [53]), Furst, Hopcroft, and Luks [45], borrowing the framework of [8], added a deterministic termination rule to the sifting process: once a set of generators of G plus the quotient of every pair of coset representatives in $\bigcup_{i=1}^{m} T_i$ have been sifted, the coset tables are complete. This observation turned all the Las Vegas algorithms given or announced in [8] into deterministic. Moreover, [45] also applied the derandomized "tower of groups" method to the stabilizer chain of a given permutation group G (G_i consists of those elements of $G \leq \mathrm{Sym}(\Omega)$ that fix the first i elements of Ω) with the far reaching consequence that *every (known) permutation group is recognizable* [45]. It also follows immediately that *every (known) permutation group is samplable.*

It was also "discovered" that in the context of the stabilizer chain, sifting had already long been known in computational group theory by Charles Sims [78,79], with a more efficient termination rule involving "strong generators." Sims, however, did not provide a proof of polynomiality of his termination rule; the proof came in a short 1981 note by Knuth which a decade later turned into a (long) paper [59].

3.5 Ignorance Is Bliss

My failure to recognize a deterministic termination rule for the "tower of groups" method had some benefits to complexity theory.

The first and smallest was the term "Las Vegas algorithm."

A more important by-product was a connection between sampling and counting. An earlier result showed that Graph Isomorphism was equivalent to counting automorphisms/isomorphisms of graphs [7,68].

Now the "tower of groups" method yielded sampling of all members of the subgroup chain, including the automorphism group. Once the coset tables filled up, we know the order of each group in the chain:

$$|G_j| = \prod_{i=j+1}^{m} |G_{i-1} : G_i| = \prod_{i=j+1}^{m} |T_i|. \tag{1}$$

So here was a case where the solution of the isomorphism counting problem was achieved *through solving random generation first.*

This made me wonder if something like this could hold in a broader context. While the exactness of counting could harly be expected for $\#P$-complete problems, I was led to the question whether *approximate counting* could be achieved given approximate sampling for $\#P$-hard counting problems. These two indeed turned out to be equivalent in a number of important contexts; my favorite examples were perfect matchings and k-colorings. The key was a certain reduction of these problems to their smaller instances ("self-reducibility," as it later came to be called), foreshadowed by Eq. (1), cf. [60, p. 33]. I outlined the method, which later became known as "JVV-reduction" ([55], cf. [60]), in two talks at the University of Waterloo in November 1979, but never published the result[1].

Meanwhile, most importantly, randomization became bread and butter for me, a recurring feature of my work. The cases most relevant to complexity theory were the introduction of *random self-reducibility* [10] (exploiting the homogeneity of a group, albeit an abelian one), the invention of *Arthur-Merlin games* (public-coin interactive proofs, Section 4.3), and the MIP = NEXP theorem [19]. Rapid improvements of the technical details of the latter led to the PCP theorem [6]. One of these technical details was the "low-degree test" for multivariate polynomials, which, along with [33], started the field of *property testing.* Locally testable/correctable codes also have part of their roots in the technical details of [19].

Groups are particularly amenable to randomization, as demonstrated in the subsequent development of the matrix group membership problem.

3.6 Permutation Groups in NC

While the polynomial-time algorithms of Sims, Furst-Hopcroft-Luks, Knuth to test membership in a given permutation group are entirely elementary, subsequent developments were characterized by heavy reliance on group theory. The proof of the following result illustrates this trend.

Theorem 2 ([23]). *Testing membership in a given permutation group is in NC.*

[1] The word apparently got around, though. In autumn 1984 Leslie Valiant, who, with his coauthors, rediscovered this phenomenon, called me by phone and asked if I had "formalized" it. After brief miscommunication, the call ended abruptly, and the fact that groups had been a source of motivation for this connection did not become widely known - although Lovász appears to have made the connection [60, p. 33].

The NC algorithm given in [23] maps out the normal structure of the group in great detail, finding all composition factors and much more, before it can answer the basic membership query.

4 Matrix Group Membership

By the mid-80s, the polynomial-time theory of permutation groups was well understood (cf. [65]). Two highlights should be mentioned: composition chains were found by Luks [63] and Sylow subgroups by Kantor [57] in polynomial time. The next task was to replicate this theory for matrix groups. This project started with the 1984 paper [29].

4.1 Black-Box Groups

First, [29] put the problem in the genaral framework of "black-box groups." A *black-box group* G is a finite group to which we have access through an oracle as follows.

The group oracle holds (a) a set $W \subseteq \{0,1\}^n$ of "valid strings:" (b) a surjection $\gamma : W \to G$ (the string $x \in \{0,1\}^n$ *encodes* the group element $\gamma(x)$) (c) a "multiplication function" $\mu : W \times W \to W$ and an "inversion function" $\iota : W \to W$ such that $(\forall x, y \in W)(\gamma(x)\gamma(y) = \gamma(\mu(x,y))$ and $\gamma(\iota(x)) = \gamma(x)^{-1}$. We are permitted the following queries to the oracle: given $x, y \in W$, tell $\mu(x,y)$ and $\iota(x)$ (multiplication and inversion); and given $x \in W$, tell whether or not $\gamma(x) = 1$ (identity query). G is "given" by a list of "generators," i. e., by a list of codewords $x_1, \ldots, x_s$ such that $\{\gamma(x_1), \ldots, \gamma(x_s)\}$ is a set of generators of G.

We call n the *encoding length* of G; we have $|G| \leq 2^n$.

Note that queries about strings $x \notin W$ are illegal; in particular, no membership test in W is provided.

We say G is a *uniquely encoded black-box group* if the function γ is bijective. Quantum complexity theory adopted the black-box group concept in the meaning of uniquely encoded black-box groups. While this is necessary because of the reversibility of quantum computation, it loses much of the theory which critically depends on the nonuniqueness of encoding through the following observation.

Observation 3 ([29]). *Let G be a black-box group and N a recognizable normal subgroup of G. Then G/N can be treated as a black-box group.*

Indeed, the oracle will be the same, except that identity queries will be replaced by membership queries to N.

We illustrate the significance of this observation in Section 4.6. We note that the theory would remain essentially intact if we required *uniform encoding* (each set $\gamma^{-1}(g)$ has the same size, where $g \in G$), but even this uniform ambiguity is not acceptable in quantum computation, while in the general theory, we have yet to find any use of the uniformity assumption.

4.2 Nondeterministic Complexity of Membership

The first question, raised in [29], was the nondeterministic complexity of membership in matrix groups over finite fields. It is not evident that this problem is in NP, and to this day we cannot prove that it is in coNP (although we are close).

The NP question was resolved in [29] in a very general form. A *straight-line program* (SLP) in a group G given by a set S of generators is a list $(a_1, \dots, a_m)$ of elements of G such that each a_i is either (a) itself a generator, or (b) is obtained as $a_i = a_j a_k$ for some $j, k < i$, or (c) $a_i = a_j^{-1}$ for some $j < i$. We say that this SLP reaches a_m from S in m steps.

Theorem 4 ([29]). *Every element of a finite group G can be reached from any set of generators in $\leq (1 + \log |G|)^2$ steps.*

It is an immediate consequence that membership in a black-box group is in NP (relative to the oracle): the straight-line program is the witness; its length is $O(n^2)$ where n is the encoding length.

The construction is a nonabelian generalization of the method of repeated squares; we call it the "cube doubling" method.

The sampling algorithm of Theorem 1 works in the generality of black-box groups. The idea is an algorithmic realization of "cube doubling" via random walks.

4.3 Nondeterministic Complexity of Nonmembership. Arthur-Merlin Games, Almost-NP

While Theorem 4 puts matrix group membership in NP, the statement that "matrix group membership is in coNP" has the status of **"almost theorem:"** it follows from the **Short Presentation Conjecture** stated in [29]. The conjecture says that *every finite simple group G has a presentation (in terms of generators and relations) of bitlength* polylog $|G|$ *which is efficiently computable from the standard name of G.* This conjecture has been verified for all classes of finite simple groups except for the "rank-1 Ree groups" ${}^2G_2(q)$ (defined over the field $\mathbb{F}_q$ where q is an odd power of 3) [80,20,54].

The "almost theorem" that matrix group nonmembership is in NP was soon complemented by the theorem that matrix group nonmembership is in "almost NP."

Theorem 5 ([9]). *Matrix group nonmembership belongs to AM.*

Here AM is the class of languages recognizable by a public-coin interactive proof where the verifier (Arthur) flips coins and the prover (Merlin) responds. This class was introduced in [9] as a randomized extension of NP. While the motivation came from the application to the complexity of matrix group nonmembership, [9] also began the complexity theoretic study of this class and the related Arthur-Merlin hierarchy. It was noted, in particular, that AM $\subseteq$ Π_2. Another notable member of this hierarchy is MA (Merlin moves first, Arthur responds); this is the class of languages with "publishable proof" of membership. (Merlin's written proof remains verifiable by the Arthurs of posterity.) It was shown that MA $\subseteq$ AM and therefore MA $\subseteq \Pi_2 \cap \Sigma_2$. Recently, analogues of MA have been studied in various models such as multiparty communication complexity [48] and quantum complexity [1].

It should be mentioned that the first private-coin interactive proof [49] was also about groups, although abelian ones (the multiplicative group of the ring $\mathbb{Z}/n\mathbb{Z}$).

The notion of "almost-NP" was formalized in [9]. For a relativizable complexity class $\mathcal{C}$, let *almost-$\mathcal{C}$* consist of those languages L for which $L \in \mathcal{C}^A$ for almost all oracles A. Almost-P = BPP by a result of Bennett and Gill [32]. In [9] I claimed that almost-NP = AM and that this fact follows along the lines of the Bennett-Gill argument. Soon after the publication of [9] I realized a fault in my argument; for it to work, one would need a "super-Arthur" [24] capable of flipping exponentially many coins. For a polynomially bounded Arthur to simulate super-Arthur, one would need to produce a pseudorandom string of length N from a seed of length polylog(N) that fools DNF formulas. I thought that polylog-wise independent variables (cf. [3]) should do the trick and would in fact beat all bounded-depth Boolean circuits. This problem later became known as the "Nisan-Linial conjecture" [69] and was recently confirmed, in the form I needed it (but not in the more specific form asked by [69], which turned out to be false), in a *tour de force* by Mark Braverman [36].

The almost-NP = AM statement, however, was confirmed already in January 1988 when, on my visit to Jerusalem, Avi Wigderson greeted me with the news of Noam Nisan's brilliant pseudorandom generator [70]. This pseudorandom generator was sufficient to supply the super-Arthur bits. Given this elegant solution, I stopped pursuing the "Nisan-Linial conjecture."

4.4 Randomization Tools

I'd like to mention three randomization tools for black-box groups.

A *random subproduct* of a list $g_1, \dots, g_k$ of group elements is a product of the form $g_1^{\epsilon_1} \dots g_k^{\epsilon_k}$ where the $\epsilon_i \in \{0,1\}$ are chosen by coin flips. It is easy to show that if $H < G$ is a proper subgroup and the g_i generate G then the probability that a random subproduct belongs to H is $\leq 1/2$. It follows that $O(\log |G|)$ random subproducts generate G with high probability, allowing us to avoid an explosion in the number of generators in the course of an algorithm. The method also allows the construction of normal closures, with the consequence that for any black-box group, we can construct the commutator chain and the descending central series and hence decide solvability and nilpotence in randomized polynomial time. Thus the rudiments of the algorithmic treatment of black-box groups have been obtained [17].

The second tool is an isoperimetric inequality that establishes a non-negligible rate of expansion for all finite groups. For $T \subseteq G$ we write T^k for the set of k-term products of members of T.

Theorem 6. *(Local expansion of groups) Let S be a set of generators of the group G; let $T = S \cup S^{-1} \cup \{1\}$. Let $D \subseteq T^d$ and let $0 < \alpha < 1/(2d+1)$ be such that $|D| \le (1 - 2\alpha d)|G|$. Then for some $g \in S$ we have $|Dg \setminus D| \ge \alpha|D|$.*

In other words, subsets D of a group expand proportionally to $1/d$ where d is the diameter of D in the word metric. This result first appeared as [12, Lemma 10.2] where it was used to put a number of questions about groups in AM. The result immediately found algorithmic applications; in particular, it became a key tool in the analysis of nearly-linear time algorithms for "small-base groups" (permutation groups such that the pointwise stabilizer of a polylogarithmic size subset of the permutation domain consists of the identity only) [18].

Theorem 6 implies that a random walk on the group will rapidly escape from small-diameter subsets. This consequence was the central ingredient of the most important randomization tool for group algorithms, Theorem 1, which establishes approximate sampling in black-box groups in polynomial time.

An improved sampling algorithm was proposed by Cooperman [43] and verified by Dixon [44]. "Product replacement," an empirically fast but unproven heuristic, was introduced by Leedham-Green and Soicher (cf. [39]) and is the method of choice in practical implementations. One should note that the use of such a heuristic is perfectly safe in a Las Vegas algorithm; but in randomized algorithms that are not Las Vegas, some concern, even practical, remains. Attempts at the analysis of this problem lead to difficult open questions about the automorphism group of the free groups [46].

An elegant version of Theorem 6 was given in [28]; a generalization of that result plays a role in differential geometry [81].

4.5 Cryptographic Barriers

The matrix group membership problem quickly runs into cryptographic obstacles.

The *constructive membership problem* requires not only to decide whether or not $g \in G$ (where $g \in U$, our ambient group that contains G) but also in the case of a positive answer, to exhibit a short straight-line program that reaches g from the given set of generators of G.

It is easy to see that for 1×1 matrices over $\mathbb{F}_q$, the constructive membership problem includes the Discrete Log problem over $\mathbb{F}_q$.

We do not know how to determine the order of a 1×1 matrix without knowing either the prime factorization of $q - 1$ or solving Discrete Log.

The membership (decision) problem for groups of 2×2 *diagonal* matrices includes the *decisional Diffie-Hellman problem*, a standard hard problem in cryptography.

So the reasonable question is: *What can we do in (randomized) polynomial time, permitting access to "number-theory oracles"* (factoring and Discrete Log)?

The surprising recent answer is that, at least in odd characteristic, there are *no other obstacles* to polynomial-time randomized membership testing; in this sense, we can say that **all obstacles are abelian**.

Theorem 7. *Let $G \leq \mathrm{GL}(n,q)$ be a given matrix group in odd characteristic (q is an odd prime power). Then membership in G can be decided and the order of G determined in randomized polynomial time with access to number-theory oracles.*

(Determining the order of groups suffices for membership testing since $g \in G$ exactly if the order of G does not change if we add g to the generators.)

Theorem 7 appears in [16] and is the culmination of the program outlined in [15]. A number of recent results in statistical group theory are required for the analysis [26,58,22,5,71], including a recent powerful analysis by Parker and Wilson [71] of an algorithm of Bray [35] to find the centralizer of an involution.

For solvable matrix groups, Luks solved the membership problem in deterministic polynomial time, necessarily relying on the number-theory oracles which are needed already in the abelian case [64]. Luks's algorithm puts no restriction on the characteristic. The proof of Theorem 7 reduces the problem to the solvable case and then calls Luks's algorithm.

4.6 The BB Filtration

The structural frame that led to Theorem 7 is given by the following chain of characteristic subgroups, introduced in [15] and known as the BB-filtration.

$$1 \leq \mathrm{Rad}(G) \leq \mathrm{Soc}^*(G) \leq \mathrm{Pker}(G) \leq G \tag{2}$$

We define the terms in this chain. $\mathrm{Rad}(G)$ is the solvable radical. Let $H = G/\mathrm{Rad}(G)$ and let $\varphi : G \to H$ be the natural surjection. Then $\mathrm{Soc}^*(G) = \varphi^{-1}(\mathrm{Soc}(H))$ where $\mathrm{Soc}(H)$, the *socle* of H, is defined as the product of the minimal normal subgroups of H. This group is the direct product of nonabelian simple groups: $\mathrm{Soc}(H) = T_1 \times \cdots \times T_m$, where the T_i are nonabelian simple. Conjugation by G permutes the set $\{T_1, \ldots, T_m\}$; thus we obtain a permutation representation $G \to S_m$. We define $\mathrm{Pker}(G)$ as the kernel of this representation.

To find the order of G, we only need to find the order of each of the four "layers" in this chain. We note that the top layer, $G/\mathrm{Pker}(G)$, is a permutation group ($\leq S_m$); the second layer, $\mathrm{Pker}(G)/\mathrm{Soc}^*(G)$ is a subgroup of $\mathrm{Out}(T_1) \times \cdots \times \mathrm{Out}(T_m)$ ($\mathrm{Out}(G)$ denotes the outer automorphism group of G) and is therefore solvable; the third layer, $\mathrm{Soc}^*(G)/\mathrm{Rad}(G)$ is a product of simple groups; and the most elusive fourth layer, $\mathrm{Rad}(G)$, is solvable.

We are able to determine the order of each of the top three layers in random polynomial time; only the handling of the radical requires number-theory oracles. Moreover, only *finding* the radical requires that we exclude characteristic 2.

Theorem 8 ([16]). *For any matrix group G over a finite field, we can determine the order of $G/\mathrm{Rad}(G)$ in random polynomial time.*

The decision version of the problem, "Does G have a quotient of order greater than a given value M without abelian normal subgroups" is in BPP, and is not known to be in R or coR. This seems to be one of the very few natural problems with this complexity status.

The following result illustrates the significance of Observation 3. The results says that a rather deep exploration of the top three layers can actually be accomplished in the generality of black-box groups, without number-theory oracles.

Theorem 9. *Let G be a given black-box group and assume a superset of the prime factors of $|G|$ is given. Then the standard names of all nonabelian composition factors of G can be listed in randomized polynomial time.*

This was the first confirmation of the philosophy that "all obstacles are abelian." For the "standard names" of finite simple groups (such as $\mathrm{PSL}(n, q)$) we refer to [42].

This result appears in [16] and relies on several papers in statistical group theory that were motivated by it and were published later [26,5,58,22].

The first step in the process to prove Theorem 9 is that we move from G to $G/\operatorname{Rad}(G)$ via Observation 3, noting that the radical is recognizable: $g \in \operatorname{Rad}(G)$ exactly if the normal closure of G is solvable. (As indicated in Section 4.4, normal closures can be constructed and solvability decided in black-box groups.)

4.7 Constructive Membership Test. Center

Under the conditions of Theorem 7, not only are we able to decide membership in G but we can do so *constuctively.* The proof of this requires considerable extra work and relies, in addition to the works cited above, on a recent algorithm from [52]. We note an important corollary.

Corollary 10. *Under the conditions of Theorem 7, the center of G can be found.*

This corollary rests on the following general principle.

Proposition 11. *Let $\varphi : G \to H$ be a homomorphism given by the images of a set of generators of G, where G and H are a black-box group and every known subgroup of H admits constructive membership testing. Then the kernel of φ can be constructed.*

This result holds with respect to randomized polynomial-time computation, with or without number-theory oracles.

Now to prove the Corollary, we only need to note that if $G \leq \mathrm{GL}(d, q)$ then G acts by conjugation on the linear span of G in the space of $d \times d$ matrices over $\mathbb{F}_q$. The kernel of this action is the center of G.

4.8 Statistical Group Theory

We mention two of the recent results in statistical group theory motivated by [15] and required for the proof of the results stated in the Section 4.6. The proof of each result is based on a detailed analysis of the list of finite simple groups [42].

Theorem 12 ([26]). *If G is a finite simple group and r is a prime then at least an $\Omega(1/\sqrt{\log |G|})$ fraction of the elements of G have order relatively prime to r.*

The applications of this result include splitting a direct product of simple groups into its factors; finding the center $Z(G)$ of a group G where $G/Z(G)$ is simple; recognizing the third level of the BB filtration ($\mathrm{Soc}^*(G)$) within the second level ($\mathrm{Pker}(G)$).

The next theorem says that we can compute the order of a finite simple group from a statistic of the orders of its elements.

Let $E = (e_1, \ldots, e_m)$ be a list of positive integers and $g \in G$ a group element. The "result of the E-test on g" is the $(0,1)$-sequence $(\epsilon_1, \ldots, \epsilon_m)$ where $\epsilon_i = 1$ exactly if $g^{e_i} = 1$. The "result of the E-test on the sequence $g_1, \ldots, g_t \in G$" is the concatenation of the results of the E-tests on each g_i.

Theorem 13 ([58,22]). *Given a prime p and a positive integer $N \geq p$, we can compute in time* polylog(N) *a sequence $E = (e_1, \ldots, e_m)$ of positive integers with the following property. Let G be a finite simple group of Lie type in characteristic p and suppose $|G| \leq N$. Then the result of the E-test on a sequence of* polylog(N) *nearly uniform random elements of G determines, with high probability, the order of G; and the order of G can be calculated in* polylog(N) *time from the data, with high probability.*

We note that there is only one infinite family of pairs of nonisomorphic simple groups of equal order: the symplectic groups $\mathrm{PSp}(2d, q)$ and the orthogonal groups $P\Omega(2d+1, q)$ for odd q. A black-box algorithm to separate these pairs, based on an analysis of Bray's algorithm for centralizers [35], was found by Altseimer and Borovik [5].

5 Prequel: Leningrad, November 4, 1978

5.1 Ten Rubles Extra

A few minutes before midnight on Friday, November 3, 1978, I boarded an overnight train at Moscow's "Leningrad Railway Station." The next day I would spend the afternoon in intense discussion with Dima Grigoriev at LOMI, the Leningrad Branch of the Mathematical Institute of the Soviet Academy of Sciences, a discussion that would profoundly affect my research.

The trip was not without risks. I did not have a permit to travel to Leningrad; the need to do so was not foreseen. Pretending to be a Soviet citizen, with limited knowledge of Russian, I traveled in a railcar foreigners were not supposed to enter. After waiting at the station for hours in vain at the end of a long line before the ticket window, I obtained my 13-ruble ticket ($13 on official exchange rate) from a courteous individual who was looking for some private business in this wholly state-owned economy, after the little ticket-window closed without having sold a single ticket. My volunteer ticket agent charged me "chervonyets sverhoo" - 10 rubles extra, still a bargain for an 8-hour trip in a sleeping car. Once in the car, I had to pretend I could hardly keep my eyelids open, to avoid having to chat, maybe share a drink of vodka, with the friendly Russian crowds on the train. Had I been discovered (uttering a single word would have sufficed), my career would have taken a rather different turn.

I was on a scientific exchange visit pursuant to an agreement between the governments of the fraternal nations of Hungary and the Soviet Union. The itinerary had to be planned years in advance; I was to visit the state universities of Moscow and Minsk.

Minsk was my first stop. One of my then recent results was an algorithm to test isomorphism of graphs with simple eigenvalues in polynomial time, and I was working on the case of bounded eigenvalue multiplicity. I learned from my colleagues in Minsk that a Leningrad mathematician by the name of Dmitry Grigoriev had announced a solution to just that problem under the additional assumption of *vertex-transitivity* (all vertices are equivalent under automorphisms). I found this extra condition unnatural ("symmetry can only make the isomorphism problem easier") and decided that I must meet Grigoriev.

There was no group theory in Dima's solution; it used vertex-transitivity only in the naive sense ("all vertices look alike" so no vertex-invariants can distinguish them). By the end of the day, I understood the linear algebra involved, and recognized the combinatorial difficulty in extending the solution to the general case. I left confident that sooner rather than later that combinatorial difficulty would fall, and Dima entrusted the further fate of that problem to me. Within hours of our farewell, I was on my way to Moscow, the return ticket having been secured by my friend Ruvim Gurevich. Having survived this sidetrip without incident[2], a few days later I was back home in Budapest, merrily preparing for my first trip to the US.

5.2 Cylinder Intersection

Within days of my meeting with Dima it became clear that the automorphism group of a graph with bounded multiplicity of eigenvalues can be represented as the intersection of certain "cylindric subgroups with small base" of a direct product of small groups. Here a *cylindric subgroup* of the group $H_1 \times \cdots \times H_k$ is a group of the form $B \times \prod_{j \notin J} H_j$ where $J \subseteq \{1, \dots, k\}$ and $B \leq \prod_{j \in J} H_j$. The group B is the "base" of the cylinder. "Small" means polynomially bounded size. So for the first time, the key to a case of Graph Isomorphism was a problem in computational group theory.

In Budapest I announced a solution to this problem and at the end of November I left for New York believing that the case of bounded eigenvalue multiplicity had been conquered.

5.3 Las Vegas in Montreal

I found the graphs with bounded color multiplicity as a handy model that showed the same combinatorial difficulty without the linear algebra. Indeed, for this class of graphs, the automorphism group is the intersection of cylinders with *bounded* bases.

[2] There was more thrill to this adventure than this brief account could indicate; for details, see [13].

On a visit to MIT in spring 1979, I tried to use this model to explain the solution to Gary Miller. I could not. There was no solution.

After several agonizing weeks, in a flash on a sunny day at a meeting in Montreal in June 1979, the "tower of groups" solution struck me, first as an attempt to put this case of Graph Isomorphism in $\mathrm{NP} \cap \mathrm{coNP}$, then in $\mathrm{ZPP} = \mathrm{R} \cap \mathrm{coR}$. The solution also resolved the general case of cylinder intersections [8] and thus completed the polynomial-time (then Las Vegas) isomorphism test for graphs with bounded eigenvalue multiplicity, also derandomized by the subsequent [45] termination rule [21].

More importantly, and not foreseen in Leningrad on November 4, 1978, the method linked the Graph Isomorphism problem to group theory, initiated the complexity analysis of algorithmic problems for finite groups, and linked groups to a randomized complexity class - precipitating the developments described in this paper.

My 10 rubles were well spent.

References

1. Aaronson, S., Kuperberg, G.: Quantum versus classical proofs and advice. Theory of Computing 3, 129–157 (2007)
2. Aaronson, S., Le Gall, F., Russell, A., Tani, S.: The one-way communication complexity of group membership. arXiv:0902.3175v2
3. Alon, N., Babai, L., Itai, A.: A fast and simple randomized parallel algorithm for the maximal independent set problem. J. of Algorithms 7, 567–583 (1986)
4. Alon, N., Lubotzky, A., Wigderson, A.: Semi-direct product in groups and zig-zag product in graphs: connections and applications. In: Proc. 42nd FOCS, pp. 630–637. IEEE Computer Soc., Los Alamitos (2001)
5. Altseimer, C., Borovik, A.V.: Probabilistic recognition of orthogonal and symplectic groups. In: Groups and Computation III, deGruyter, pp. 1–20 (2001)
6. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof Verification and the Hardness of Approximation Problems. J. ACM 45(3), 501–555 (1998)
7. Babai, L.: On the isomorphism problem. Manuscript, 10 pages (1977)
8. Babai, L.: Monte Carlo algorithms in graph isomorphism testing. Université de Montréal Tech. Rep., DMS 79-10, 42 pages (1979)
9. Babai, L.: Trading group theory for randomness. In: Proc. 17th STOC, pp. 421–429. ACM, New York (1985)
10. Babai, L.: Random oracles separate PSPACE from the polynomial-time hierarchy. Information Processing Letters 26, 51–53 (1987/1988)
11. Babai, L.: Local expansion of vertex-transitive graphs and random generation in finite groups. In: Proc. 23rd STOC, pp. 164–174. ACM, New York (1991)
12. Babai, L.: Bounded round interactive proofs in finite groups. SIAM J. Discr. Math. 5, 88–111 (1992)
13. Babai, L.: The forbidden sidetrip. In: Calude, C.S. (ed.) People & Ideas in Theoretical Computer Science, pp. 1–31. Springer, Heidelberg (1998)
14. Babai, L., Banerjee, A., Kulkarni, R., Naik, V.: Evasiveness and the distribution of prime numbers. In: Proc. 27th Ann. Symp. on Theoretical Aspects of Comp. Sci (STACS 2010), pp. 71–82. Schloss Dagstuhl Online Publ. (2010)

15. Babai, L., Beals, R.: A polynomial-time theory of black-box groups I. In: Groups St Andrews 1997 in Bath, I. London Math. Soc. Lect. Notes. vol. 260, pp. 30–64 (1999)
16. Babai, L., Beals, R., Seress, Á.: Polynomial-time theory of matrix groups. In: Proc. 41st ACM STOC, pp. 55–64 (2009)
17. Babai, L., Cooperman, G., Finkelstein, L., Luks, E.M., Seress, Á.: Fast Monte Carlo algorithms for permutation groups. J. Computer and System Sci. 50, 296–308 (1995); (Prelim. 23rd STOC, 1991)
18. Babai, L., Cooperman, G., Finkelstein, L., Seress, Á.: Nearly linear time algorithms for permutation groups with a small base. In: Proc. ISSAC 1991 Internat. Symp. on Symbolic and Algebraic Computation, Bonn, pp. 200–209 (1991)
19. Babai, L., Fortnow, L., Lund, C.: Nondeterministic exponential time has two-prover interactive protocols. Computational Complexity 1, 3–40 (1991); Prelim. version FOCS 1990, pp. 26–34 (1990)
20. Babai, L., Goodman, A.J., Kantor, W.M., Luks, E.M., Pálfy, P.P.: Short presentations for finite groups. J. Algebra 194, 79–112 (1997)
21. Babai, L., Grigor'ev, D.Y., Mount, D.M.: Isomorphism of graphs with bounded eigenvalue multiplicity. In: 14th ACM STOC, pp. 310–324 (1982)
22. Babai, L., Kantor, W.M., Pálfy, P.P., Seress, Á.: Black-box recognition of finite simple groups of Lie type by statistics of element orders. J. Group Theory 5, 383–401 (2002)
23. Babai, L., Luks, E.M., Seress, Á.: Permutation groups in NC. In: Proc. 19th ACM STOC, pp. 409–420 (1987)
24. Babai, L., Moran, S.: Arthur-Merlin games: A randomized proof system and a hierarchy of complexity classes. J. Computer and Sys. Sci. 36, 254–276 (1988)
25. Babai, L., Nikolov, N., Pyber, L.: Product growth and mixing in finite groups. In: Proc. 19th Ann. Symp. on Discrete Algorithms (SODA 2008), pp. 248–257. ACM-SIAM (2008)
26. Babai, L., Pálfy, P.P., Saxl, J.: On the number of p-regular elements in simple groups. LMS J. Computation and Math. 12, 82–119 (2009)
27. Babai, L., Shalev, A.: Recognizing simplicity of black-box groups and the frequency of p-singular elements in affine groups. In: Groups and Comp. III, deGruyer, pp. 39–62 (2001)
28. Babai, L., Szegedy, M.: Local expansion of symmetrical graphs. Combinatorics, Probability, and Computing 1, 1–11 (1992)
29. Babai, L., Szemerédi, E.: On the complexity of matrix group problems I. In: Proc. 25th FOCS, pp. 229–240. IEEE Comp. Soc., Los Alamitos (1984)
30. Barrington, D.A.: Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1. J. Comp. Sys. Sci. 38, 150–164 (1989)
31. Beals, R.: Towards polynomial time algorithms for matrix groups. In: Groups and Computation II, DIMACS, pp. 31–54 (1997)
32. Bennett, C.H., Gill, J.: Relative to a random oracle A, $P^A \neq NP^A \neq coNP^A$ with probability 1. SIAM J. Comp. 10, 96–113 (1981)
33. Blum, M., Luby, M., Rubinfeld, R.: Self-Testing/Correcting with Applications to Numerical Problems. J. Comput. Syst. Sci. 47(3), 549–595 (1993) (Prelim. 22nd STOC, 1990)
34. Bourgain, J., Gamburd, A.: Uniform expansion bound for Cayley graphs of $SL_2(\mathbb{F}_p)$. Ann. Math. 167, 625–642 (2008)
35. Bray, J.: An improved method for generating the centralizer of an involution. Arch. Math. 74, 241–245 (2000)

36. Braverman, M.: Poly-logarithmic independence fools AC0 circuits. Conf. Computational Complexity (2009) (to appear in the J. ACM)
37. Breuillard, E., Green, B., Tao, T.: Approximate subgroups of linear groups. arXiv:1005.1881v1
38. Brooksbank, P.A., Kantor, W.M.: On constructive recognition of a black box PSL(d, q). In: Groups and Computation III, pp. 95–111. deGruyter (2001)
39. Celler, F., Leedham-Green, C.R., Murray, S., Niemeyer, A.C., O'Brien, E.A.: Generating random elements of a finite group. Comm. Alg. 23, 4931–4948 (1995)
40. Cai, J., Fürer, M., Immerman, N.: An optimal lower bound on the number of variables for graph identification. Combinatorica 12, 389–410 (1992)
41. Conder, M., Leedham-Green, C.R., O'Brien, E.A.: Constructive recognition of PSL$(2, q)$. Trans. Amer. Math. Soc. 358, 1203–1221 (2006)
42. Conway, J.H., Curtis, R.T., Norton, S.P., Parker, R.A., Wilson, R.A.: ATLAS of Finite Groups. Clarendon Press, Oxford (1985)
43. Cooperman, G.: Towards a practical, theoretically sound algorithm for random generation in finite groups. arXiv:math/0205203
44. Dixon, J.D.: Generating random elements in finite groups. Electronic J. Combinatorics, 15/1:R94 (2008)
45. Furst, M.L., Hopcroft, J., Luks, E.M.: Polynomial-time algorithms for permutation groups. In: Proc. 21st FOCS, pp. 36–41. IEEE C.S, Los Alamitos (1980)
46. Gamburd, A., Pak, I.: Expansion of product replacement graphs. Combinatorica 26, 411–429 (2006)
47. The GAP Group: GAP – Groups, Algorithms, and Programming. Version 4.4. Aachen–St. Andrews (2005), http://www.gap-system.org
48. Gavinsky, D., Sherstov, A.: A separation of NP and coNP in multiparty communication complexity. Theory of Computing 6, 227–245 (2010)
49. Goldwasser, O., Micali, S., Rackoff, C.: The knowledge complexity of interactive proof-systems. In: Proc. 17th STOC, pp. 291–304 (1985)
50. Gowers, W.T.: Quasirandom groups. Combinatorics, Probability and Computing 17(3), 363–387 (2008)
51. Helfgott, H.A.: Growth and generation in $SL_2(\mathbb{Z}/p\mathbb{Z})$. Ann. Math. 167, 601–623 (2008)
52. Holmes, P.E., Linton, S.A., O'Brien, E.A., Ryba, A.J.E.A., Wilson, R.A.: Constructive membership in black-box groups. J. Group Theory 11, 747–763 (2008)
53. Hopcroft, J.: Recent directions in algorithmic research. In: Deussen, P. (ed.) GI-TCS 1981. LNCS, vol. 104, pp. 123–134. Springer, Heidelberg (1981)
54. Hulpke, A., Seress, Á.: Short presentations for three-dimensional unitary groups. J. Algebra 245, 719–729 (2001)
55. Jerrum, M., Valiant, L.G., Vazirani, V.V.: Random generation of combinatorial structures from a uniform distribution. Theor. Comp. Sci. 43, 169–188 (1986)
56. Kahn, J., Saks, M., Sturtevant, D.: A topological approach to evasiveness. Combinatorica 4, 297–306 (1984)
57. Kantor, W.M.: Sylow's theorem in polynomial time. J. Computer Sys. Sci. 30, 359–394 (1985)
58. Kantor, W.M., Seress, Á.: Black box classical groups. Mem. AMS 149, 708:viii+168 (2001)
59. Knuth, D.E.: Notes on efficient representation of perm groups. Combinatorica 11, 33–43 (1991)
60. Lovász, L.: Random Walk on Graphs: A Survey. In: Combinatorics, Paul Erdős is 80, Vol. 2. Bolyai Society Mathematical Studies 2, Budapest, pp. 353–398 (1995)

61. Lubotzky, A., Phillips, R., Sarnak, P.: Ramanujan graphs. Combinatorica 8, 261–278 (1988)
62. Luks, E.M.: Isomorphism of graphs of bounded valence can be tested in polynomial time. J. Comp. Syst. Sci. 25, 42–65 (1982) (Preliminary version in FOCS 1980)
63. Luks, E.M.: Computing the composition factors of a permutation group in polynomial time. Combinatorica 7, 87–99 (1987)
64. Luks, E.M.: Computing in solvable matrix groups. In: Proc. 33rd FOCS, pp. 111–120 (1992)
65. Luks, E.M.: Permutation groups and polynomial-time computation. In: Groups and Computation, DIMACS, pp. 139–175 (1993)
66. Margulis, G.A.: Explicit constructions of graphs without short cycles and low density codes. Combinatorica 2, 71–78 (1982)
67. Margulis, G.A.: Explicit group theoretic construction of combinatorial schemes and their application for the construction of expanders and concentrators. Probl. Info. Transmission 24, 39–46 (1988)
68. Mathon, R.: A note on the graph isomorphism counting problem. Inf. Proc. Letters 8, 131–132 (1979)
69. Nisan, N., Linial, N.: Approximate inclusion-exclusion. Combinatorica 10(4), 349–365 (1990)
70. Nisan, N.: Pseudorandom generators for space-bounded computation. Combinatorica 12(4), 449–461 (1992)
71. Parker, C.W., Wilson, R.A.: Recognising simplicity of black-box groups (2004) (manuscript)
72. Pyber, L., Szabó, E.: Growth in finite simple groups of Lie type. arXiv:1005.1858v1
73. Reingold, O.: Undirected connectivity in logspace. J. ACM 55(4), 1–24 (2008)
74. Reingold, O., Vadhan, S., Wigderson, A.: Entropy waves, the zig-zag graph product, and new constant-degree expanders. Ann. of Math. 155(1), 157–187 (2002)
75. Rotman, J.: An Introduction to the Theory of Groups. Springer, Heidelberg (1994)
76. Seress, Á.: Permutation Group Algorithms. Cambridge University Press, Cambridge (2003)
77. Shor, P.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Computing 26, 1484–1509 (1977)
78. Sims, C.C.: Computational methods in the study of permutation groups. In: Computational problems in abstract algebra (Oxford, 1967), pp. 169–183. Pergamon Press, New York (1970)
79. Sims, C.C.: Computation with permutation groups. In: Proc. Second ACM Symp. on Symbolic and Algebraic Manipulation, pp. 23–28. ACM, New York (1971)
80. Suzuki, M.: On a class of doubly transitive groups. Ann. Math. 75, 105–145 (1962)
81. Žuk, A.: On an isoperimetric inequality for infinite finitely generated groups. Topology 39, 947–956 (2000)

On Maltsev Digraphs

Catarina Carvalho[1], László Egri[2], Marcel Jackson[3], and Todd Niven[3,⋆]

[1] School of Physics, Astronomy and Mathematics, University of Hertfordshire, UK, and Centro de Álgebra da Universidade de Lisboa, Portugal
ccarvalho@cii.fc.ul.pt
[2] School of Computer Science, McGill University, Canada
legri1@cs.mcgill.ca
[3] Department of Engineering and Mathematical Sciences, La Trobe University, Australia
{m.g.jackson,t.niven}@latrobe.edu.au

Abstract. We study digraphs preserved by a Maltsev operation, Maltsev digraphs. We show that these digraphs retract either onto a directed path or to the disjoint union of directed cycles, showing that the constraint satisfaction problem for Maltsev digraphs is in logspace, L. (This was observed in [19] using an indirect argument.) We then generalize results in [19] to show that a Maltsev digraph is preserved not only by a majority operation, but by a class of other operations (e.g., minority, Pixley) and obtain a $O(V_G^4)$-time algorithm to recognize Maltsev digraphs. We also prove analogous results for digraphs preserved by conservative Maltsev operations which we use to establish that the list homomorphism problem for Maltsev digraphs is in L. We then give a polynomial time characterisation of Maltsev digraphs admitting a conservative 2-semilattice operation. Finally, we give a simple inductive construction of directed acyclic digraphs preserved by a Maltsev operation.

1 Introduction

The study of relational structures and, in particular, digraphs preserved by certain operations from universal algebra became extremely important during the last decade. The main driving force behind this is the algebraic constraint satisfaction problem (CSP) dichotomy conjecture, which states that a constraint satisfaction problem CSP(**B**) is tractable if the relational structure **B** is preserved by a weak-near-unanimity (weak-NU) operation, and is NP-complete otherwise [5,6,20]. Generalizing the dichotomy theorem of Hell and Nešetřil [16], the conjecture has been established for digraphs with no sinks and no sources by showing that such digraphs are very structured, in fact, they retract onto a disjoint

⋆ The first author was supported by grants SFRH/BPD/26216/2006 and ISFL-1-143 of CAUL financed by FCT and FEDER. The second author was supported by the National Sciences and Engineering Research Council of Canada (NSERC). The third and fourth authors were supported by ARC Discovery Project DP1094578.

union of directed cycles [3]. Other results relating the complexity of CSPs on digraphs to the existence of operations that preserve the digraph can be found, for example, in [2,1].

Once the tractability of a CSP is established, one also wishes to know the fine-grained complexity of that CSP, i.e. is the CSP in some subclass of P, such as L or NL? To establish the membership of a CSP in a complexity class inside P it is important to study the structure of relational structures (and digraphs) preserved by operations which are more "restrictive" than weak-NU operations, i.e. operations that imply the presence of a weak-NU operation. Two important results in this direction, which we will invoke, are that if a relational structure is preserved by a majority operation, then the corresponding CSP is in the complexity class NL [9]; and if CSP($\mathbf{B}$) is definable in Datalog and $\mathbf{B}$ is preserved by a Maltsev operation, then CSP($\mathbf{B}$) is in L [10,12].

We study the structure of digraphs preserved by a Maltsev operation, that we call *Maltsev digraphs*. We show that these digraphs retract either onto the disjoint union of directed cycles or to a directed path. This gives a direct proof that the corresponding CSP is in constant width symmetric Datalog and therefore in L. (Membership of these CSPs in symmetric Datalog, without the constant width guarantee, was independently shown by Kazda [19], however, his proof is rather indirect.) We then generalise other results in [19] to show that a Maltsev digraph is preserved not only by a majority polymorphism, but also by a class of polymorphism obeying certain restrictions (e.g. minority, Pixley). We also extend the results to the conservative setting, i.e. we show that a conservative Maltsev digraph is preserved by a class of conservative polymorphisms.

A generalization of the rectangularity [4] property of digraphs is introduced. We call this rectangularity *total rectangularity*, and we establish that a digraph is preserved by a Maltsev operation iff it is totally rectangular. Similarly, we show that a digraph is preserved by a conservative Maltsev operation iff it is *universally rectangular*, a specific form of total rectangularity.

We apply our results to the list homomorphism problem, LHOM, for directed graphs. While the complexity of LHOM for undirected graphs is completely understood [13], for the directed version, the only result known is a P vs. NP dichotomy [17]. We show that LHOM for Maltsev digraphs is in L.

It is shown that a digraph preserved by a Maltsev operation is also preserved by a conservative 2-semilattice operation iff the digraph satisfies a certain combinatorial property. We note that if a structure is preserved by a 2-semilattice operation, then CSP($\mathbf{B}$) is in Datalog and therefore in P [18]. We also characterise Maltsev digraphs preserved by a conservative 2-semilattice operation and show that these digraphs can be recognised in NL.

Finally, an inductive construction of directed acyclic graphs preserved by a Maltsev operation is given. The main motivation behind this construction is that we suspect that extending this construction to n-permutable digraphs (2-permutable digraphs are precisely the Maltsev digraphs [15]) could make progress towards identifying all list homomorphism problems for digraphs in L. We note that in [13], an inductive construction of "conservative" n-permutable graphs is key to the identification of all graphs whose LHOM is in L.

2 Preliminaries

2.1 Algebra

We describe the algebraic definitions for digraphs, however, note that these definitions are straightforward to generalize to relational structures. Let $G = (V_G, E_G)$ and $H = (V_H, E_H)$ be digraphs. A *homomorphism* from G to H is a map f from V_G to V_H, such that for every edge $(u, v) \in E_G$ we have that $(f(u), f(v))$ is an edge in H, i.e. $(f(u), f(v)) \in E_H$. A digraph G is called a *core* if every homomorphism from G to itself is an automorphism, i.e. a permutation on V_G. Let G' be a subgraph of G. We say that G retracts onto G' if there is a homomorphism $h : G \to G'$ such that h is the identity map on G'. For a digraph H, we can then define $\mathrm{CSP}(H)$ as the class of all digraphs that admit a homomorphism to H.

An n-ary operation on a set A is a function $f : A^n \to A$. Given a digraph G and an n-ary operation f on V_G, we say that f *preserves* G, or that f is a *polymorphism* of G, if for any n edges $(u_1, v_1), \ldots, (u_n, v_n) \in E_G$ (not necessarily distinct), the pair $(f(u_1, \ldots, u_n), f(v_1, \ldots, v_n)) \in E_G$. For an n-ary operation f, we write $f(x_1, \ldots, x_n) \approx f(y_1, \ldots, y_n)$ if $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n)$ for all possible values of the x_i, y_i, $i = 1, \ldots, r$. A ternary operation m is *Maltsev* if it satisfies $m(x, x, y) \approx m(y, x, x) \approx y$, *Pixley* if it satisfies $m(x, x, y) \approx m(y, x, x) \approx m(y, x, y) \approx y$, *majority* if it satisfies $m(x, x, y) \approx m(x, y, x) \approx m(y, x, x) \approx x$, and *minority* if it satisfies $m(x, x, y) \approx m(x, y, x) \approx m(y, x, x) \approx y$. A binary operation $*$ is *2-semilattice* if it satisfies $x * x \approx x$, $x * y \approx y * x \approx x * (x * y)$.

2.2 Graph Theory

Since all graphs in the rest of the paper are directed, we use the terms graph and digraph interchangeably. For a natural number n we write $[n] = \{1, 2, \ldots, n\}$. An *oriented path* is a sequence of, not necessarily distinct, vertices $v_1, \ldots, v_n$ such that for every $i \in [n-1]$, either (v_i, v_{i+1}) (*a forward edge*) or (v_{i+1}, v_i) (*a backward edge*) is an edge. We use the terms path and oriented path interchangeably. A *cycle* is an oriented path with starting point v_1 and endpoint v_m such that either (v_m, v_1) or (v_1, v_m) is an edge. The *net length* of a path P, *net*(P), is the number of forward edges minus the number of backward edges in P. A *(reverse) dipath* is a sequence of, not necessarily distinct, vertices $v_1, \ldots, v_n$ such that for every $i \in [n-1]$, (v_i, v_{i+1}) $((v_{i+1}, v_i))$ is an edge. A *directed cycle* is a dipath $v_1, \ldots, v_n$ such that (v_n, v_1) is also an edge. For a (reverse) dipath P, we let $len(P)$ denote the number of edges in P. We use the term *simple* dipath or (directed) cycle to indicate that all vertices of the dipath or (directed) cycle are distinct.

A *component* of digraph G is a maximal subgraph, H, of G such that for every pair of vertices $u, v \in V_H$, there is an oriented path from u to v. A digraph with one component is said to be *connected.* A digraph is a *directed acyclic graph* (DAG) if it contains no directed cycles. A DAG G is *layered* if there exists $q \in \mathbb{N}$

such that the vertices of G can be partitioned into q levels $L_0, \ldots, L_{q-1}$, such that any edge of G goes from L_i to L_{i+1}, for some $i = 0, \ldots, q-2$.

Let G be a digraph, and x a vertex of G. We define $x^{+1} = \{y \in V_G : (x,y) \in E_G\}$, and $x^{-1} = \{y \in V_G : (y,x) \in E_G\}$. We call a vertex v a *source* if $v^{-1} = \varnothing$, and a *sink* if $v^{+1} = \varnothing$. If u and v are vertices of G, $u \xrightarrow{k} v$ denotes the existence of a dipath from u to v of length k; $u \rightarrow v$ denotes $u \xrightarrow{1} v$.

3 Retracts of Maltsev Digraphs

Definition 1 (totally rectangular). *A digraph G is k-*rectangular *if the following implication holds for all vertices x, y, u, v:*

$$x \xrightarrow{k} u \ \ \& \ \ y \xrightarrow{k} u \ \ \& \ \ y \xrightarrow{k} v \Rightarrow x \xrightarrow{k} v.$$

A digraph is rectangular *if it is 1-rectangular, and* totally rectangular *if it is k-rectangular for every $k \in \mathbb{N}$.*

It is not hard to verify that a Maltsev digraph must be totally rectangular, but in Section 4 (see Corollary 1) we show that the two properties are equivalent.

Example 1. The digraph in Fig. 1 is rectangular but not 2-rectangular. While the digraph in Fig. 4 is totally rectangular.

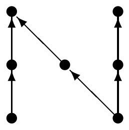


Fig. 1. A rectangular digraph that is not 2-rectangular

We now state the main result of this section.

Theorem 1. *Let G be a totally rectangular digraph. If G is acyclic then G retracts onto a simple dipath. Otherwise G retracts onto the disjoint union of simple directed cycles.*

The proof of Theorem 1 is a direct consequence of Lemma 4 below. We begin with some definitions and simple observations.

Lemma 1. *Let G be a digraph. Then G is layered iff for every pair of vertices u, v in G, and any pair of oriented paths P and Q from u to v, it holds that $net(P) = net(Q)$.*

Definition 2. *Let G be a digraph that contains a directed cycle. Let C be a shortest directed cycle in G and assume it has length m. We say that G is* inconsistent *if there exist two vertices u, v in G such that, there are two different oriented paths of net lengths ℓ_1 and ℓ_2 from u to v such that $\ell_1 \not\equiv \ell_2 \mod m$. Otherwise we say that G is* consistent.

Proposition 1. *Let G be a digraph that contains a directed cycle. Let C be a shortest directed cycle in G. Then G retracts onto C iff G is consistent.*

Lemma 2. *Let G be a totally rectangular digraph and u, v be vertices in G. Let P and Q be two dipaths in G from u to v, such that $len(P) > len(Q)$. Set $k = len(P)$, $\ell = len(Q)$, and $d = k - \ell$. Then one of the following two cases occurs:*

1. *If $2\ell > k$, then G contains vertices u', v' and dipaths P', Q' from u' to v' with the following property: $len(P') = \ell$, $len(Q') = 2\ell - k$, and $len(P') - len(Q') = d$;*
2. *If $2\ell \leq k$, then G contains a directed cycle of length d.*

Proof. See Fig. 2. In the first case, let u' be the vertex of P such that the subpath $P_{u'v}$ of P from u' to v has length ℓ. Let v' be the vertex of P such that the subpath $P_{uv'}$ of P from u to v' has length ℓ. Applying the ℓ-rectangularity of G to $P_{u'v}$, Q, and $P_{uv'}$, we obtain the desired dipath P' with $len(P') = \ell$. The other required dipath Q' is the subpath of P from u' to v'. Since $k = 2\ell - len(Q')$ we have that $len(Q') = 2\ell - k$, and $len(P') - len(Q') = \ell - (2\ell - k) = k - \ell = d$.

In the second case, the two paths P' and Q' form a cycle of length $\ell + (k - 2\ell) = d$ because $2\ell \leq k$.

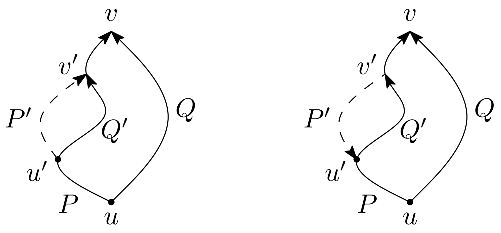


Fig. 2. Case 1 (left) and Case 2 (right) of Lemma 2

Lemma 3. *Let G be a totally rectangular digraph and $u, v \in V_G$. Let P_1 and P_2 be oriented paths in G from u to v. Assume that $net(P_1) > net(P_2)$, and set $d = net(P_1) - net(P_2)$. Then there are vertices $s, t \in V_G$ and dipaths Q_1 and Q_2 in G from s to t, such that $len(Q_1) - len(Q_2) = d$.*

Lemma 4. *Let G be a connected totally rectangular digraph. If G is a DAG then G retracts onto a simple dipath. Otherwise G retracts onto a simple directed cycle.*

Proof. Assume first that G is a DAG. We claim that G must be layered. Assume, for a contradiction, that G is not layered. By Lemma 1, there exist $u, v \in V_G$ and oriented paths P and Q from u to v, such that $net(P) \neq net(Q)$. Using Lemma 3, we can assume that P and Q are dipaths of different length. Now we repeatedly apply Case 1 of Lemma 2 as long as it is possible, and then applying Case 2 yields a cycle, a contradiction. So G is layered.

Assume that G has levels $L_0, \ldots, L_{q-1}$. Fix vertices $s \in L_0$ and $t \in L_{q-1}$, and let O be any oriented path from s to t (such a path exists because G is connected). Applying the total rectangularity of G to appropriate subpaths of O, it is easy to see that there exists a dipath D of length $q-1$ from s to t in G. Clearly, G retracts onto D.

Suppose that G contains a directed cycle. By Proposition 1, it is enough to show that G is consistent. Assume this is not the case. Let C be a shortest directed cycle in G, and assume it has length m. Because G is inconsistent, we can find vertices $u, v \in V_G$ and oriented paths P_1 and P_2 from u to v, such that $net(P_1) \not\equiv net(P_2) \mod m$. Set $\ell_1 = net(P_1)$ and $\ell_2 = net(P_2)$. Assume w.l.o.g. that $\ell_1 > \ell_2$, and that u is a vertex of C. Note that if u is not a vertex of C, then we fix a vertex c of C and find any oriented path S from c to u. Then attaching S to P_1 and P_2 at vertex u gives us the desired oriented paths. Furthermore, we can assume that $\ell_1 - \ell_2 = d < m$, because if not, we can add C-loops from u to u to P_2 to increase its length by a multiple of m, until $\ell_1 - \ell_2 < m$. Using Lemma 3 we obtain directed paths Q_1 and Q_2 such that $len(Q_1) - len(Q_2) = d$, and then, by applying Lemma 2, we obtain a cycle of length d in G, a contradiction.

By Lemma 4, each connected component of G retracts either onto a simple dipath or to a simple directed cycle. The trivial observation that a dipath homomorphically maps to a cycle completes the proof Theorem 1.

4 Characterisations, Polymorphisms and Algorithms

4.1 Rectangular Characterisations and Other Polymorphisms

In this section we generalise a technique of Kazda [19] to characterise digraphs that admit Maltsev and conservative Maltsev polymorphisms as those which are totally rectangular and universally rectangular respectively and to provide polynomial time algorithms for recognising the relevant properties. Furthermore, we show that Maltsev digraphs also admit many other polymorphisms, and under certain conditions, they also admit conservative 2-semilattice polymorphisms.

Definition 3 (conservatively k-rectangular, universally rectangular). *We say that a graph is* conservatively k-rectangular *if it satisfies the following sentence:*

$$\left.\begin{array}{l} x \to x_1 \to \cdots \to x_{k-1} \to u \\ y \to y_1 \to \cdots \to y_{k-1} \to u \\ y \to z_1 \to \cdots \to z_{k-1} \to v \end{array}\right\} \Rightarrow \left\{\begin{array}{l} \textit{There is a path } x \to w_1 \to \cdots \to \\ w_{k-1} \to v \textit{ with } w_i \in \{x_i, y_i, z_i\} \\ \textit{for each } i. \end{array}\right. \qquad (1)$$

A graph that is conservatively k-rectangular for all $k \geq 1$ will be called universally rectangular.

Example 2. The digraph in Fig. 4 is conservatively rectangular but not conservatively 2-rectangular. While the digraph in Fig. 3 is universally rectangular.

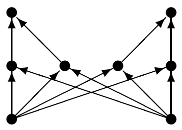


Fig. 3. A universally rectangular digraph

Definition 4. *Let G be a digraph. Define the binary relations R^- on V_G by $x\ R^-\ y$ if $x^{-1} \cap y^{-1} \neq \varnothing$. The dual relation R^+ is defined by $x\ R^+\ y$ if $x^{+1} \cap y^{+1} \neq \varnothing$.*

The relation R^+ is an equivalence relation on the set $\{x \in V_G : x^+ \neq \varnothing\}$, the set of vertices of G that are not sinks. The relation R^- is an equivalence relation on the set $\{x \in V_G : x^- \neq \varnothing\}$, the set of vertices of G that are not sources. So it makes sense to consider the respective factor graphs, this was observed in [19]. We use the notation G/R^+ to denote the graph on the R^+-classes of G. Given R^+-classes A, B, we write $A \to B$ if there is some $a \in A$ and $b \in B$ with $a \to b$. Similarly, G/R^- denotes the same construction, but using the relation R^-. Note that G/R^+ is not strictly an actual graph quotient of G, only a quotient of an induced subgraph of G. Nevertheless, we sometimes refer to it as "the quotient of G by R^+". The proof of the following lemma is routine.

Lemma 5. *Let G be a rectangular digraph and $k > 1$.*

1. *G is ℓ-rectangular for all $\ell = 1, \dots, k$ if and only if G/R^+ is ℓ-rectangular for all $\ell = 1, \dots, k-1$.*
2. *If G is conservatively ℓ-rectangular for all $\ell = 1, \dots, k$ then G/R^+ is conservatively ℓ-rectangular for all $\ell = 1, \dots, k-1$.*

For a totally rectangular graph G, define $G_0 = G$ and $G_{i+1} = G_i/R^+$, $i \geq 1$. From Lemma 5 it follows that G_i is defined for all positive integers i, and eventually G_i will either be empty or a disjoint union of directed cycles (the only situations that R^+ can be trivial). We define $G_\infty = G_k$, where k is such that $G_k = G_{k+1}$ (i.e. G_∞ is either empty or a disjoint union of directed cycles).

The next lemma is obtained by applying the Maltsev property to the columns of the premise of (1).

Lemma 6. *Let G be a digraph.*

1. *If G has a Maltsev polymorphism, then G is totally rectangular.*
2. *If G has a conservative Maltsev polymorphism, then G is universally rectangular.*

Lemma 7. *Let a, b be vertices in a totally rectangular digraph G satisfying conservative 2-rectangularity and assume that neither a nor b is a source or sink. If $a/R^+ \cap b/R^-$ is nonempty then either $b \in a/R^+$ or $a \in b/R^-$.*

Proof. Let $c \in a/R^+ \cap b/R^-$. There are vertices e, f, g, h such that $\{a, c\} \subseteq e^{-1}$, $b \in f^{-1}$, $a \in g^{+1}$ and $\{c, b\} \in h^{+1}$. However G is conservatively 2-rectangular so that there is either an edge from at least one of a, c to f or there is an edge from g to at least one of $\{b, c\}$. Then 1-rectangularity shows that either there is an edge from a to f or from g to b.

Theorem 2. *Consider a property C of digraphs defined by the existence of polymorphisms $t_1, t_2, \ldots, t_k$ (not necessarily distinct) satisfying a single equational sequence*

$$t_1(x_{1,1}, x_{1,2}, \ldots, x_{1,n_1}) \approx \cdots \approx t_k(x_{k,1}, x_{n,2}, \ldots, x_{k,n_k}) \approx x,$$

where $\{x_{1,1}, \ldots, x_{1,n_1}\} = \cdots = \{x_{k,1}, \ldots, x_{k,n_k}\}$ and $x \in \{x_{1,1}, \ldots, x_{1,n_1}\}$. The following statements are true provided that the equation $x \approx y$ does not follow from C.

1. *Let G be a totally rectangular digraph. Then G has property C if and only if G_∞ has property C.*
2. *Let G be universally rectangular. Then G has property C with each t_i conservative if and only if G_∞ has property C with each of the t_i conservative.*

The same conclusions can be made without the requirement that $\approx x$ be included in the equational sequence, and if the polymorphisms are required to be idempotent.

Proof. Our proof is very similar to the main proof in [19]; we use Lemma 5 rather than the assumption of the Maltsev property directly. We focus only on the conservative case (not considered in [19]), as the non-conservative case is obtained by following this proof and missing some steps. We give only a sketch here.

It is easy to see that if G has conservative property C then so does G/R^+ by defining $t_i(x_1/R^+, \ldots, x_n/R^+) = t_i(x_1, \ldots, x_n)/R^+$. Thus, it suffices to show that if G/R^+ satisfies some conservative property C then so does G. This uses only total rectangularity (to ensure that successive quotients are well defined).

The reverse direction is shown by backward induction over successive quotients by R^+: essentially, provided G/R^+ has polymorphisms t_i witnessing property C, then so does G. This part of the argument is mostly identical to that given in [19] (in the case of majority polymorphisms), except Lemma 7 is invoked to guarantee a conservative choice of the polymorphisms.

It is useful to note that instead of explicit use of universal rectangularity, this argument used only the fact that on each successive quotient by R^+, both rectangularity and the conclusion of Lemma 7 hold.

Some instances of polymorphisms satisfying the conditions of Theorem 2 are majority, Maltsev and Pixley. In these cases G_∞ always has the desired polymorphism, giving the following corollary.

Corollary 1. *Let G be a digraph.*

1. *G admits a (conservative) Maltsev polymorphism iff it admits a (conservative) Pixley operation iff it is totally (universally) rectangular.*
2. *If G is totally (universally) rectangular then G admits a (conservative) minority polymorphism and a (conservative) majority polymorphism.*

Remark 1. The first part of Corollary 1 strengthens the result given in Lemma 4 of [11], for the case of digraphs. The relational clone $\langle \mathbf{B} \rangle$ of a structure $\mathbf{B}$ is the set of all relations that can be expressed with primitive positive first-order formulas (i.e. only existential quantification, conjunction, and equality is allowed) from $\mathbf{B}$. When we restrict [11, Lemma 4] to digraphs, it can be stated as follows: A digraph G is preserved by a Maltsev operation iff every binary relation in $\langle G \rangle$ is rectangular. It is easy to see and well-known that every binary relation in $\langle G \rangle$ can be expressed as $B_G(S, a, b) = \{(h(a), h(b)) | h$ is a homomorphism from S to $G\}$ for a structure S with two distinguished vertices a and b. Then Corollary 1 implies that for a digraph G to be preserved by a Maltsev operation it is enough to require that only those binary relations in $\langle G \rangle$ that can be expressed as $B_G(S, a, b)$, where S is a directed path with initial vertex a and terminal vertex b, are rectangular.

The above corollary yields an algorithm for verifying if a graph has a Maltsev (or Pixley) polymorphism. Indeed, the rectangularity of a digraph is equivalent to the following property of its adjacency matrix: when two rows (or two columns) share a common 1 they are identical. On an n-vertex digraph this property may be verified in $O(n^3)$ steps. A digraph has a Maltsev polymorphism if and only if each (of at most n) successive quotient by R^+ is rectangular, with the process stopping once there are no R^+-classes of size more than 1 (which happens after at most n quotients). Overall this takes $O(n^4)$ steps (quadratic in terms of the size of the adjacency matrix).

Universal rectangularity (equivalently, the existence of a conservative Maltsev polymorphism) can also be verified in polynomial time by verifying total rectangularity and conservative 2-rectangularity at each successive quotient by R^+. In fact, the proof of Theorem 2 is sufficiently constructive to construct the desired polymorphisms (when they exist): simply work backwards from their definition of G_∞.

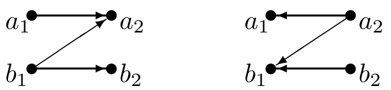
4.2 Conservative 2-Semilattice Polymorphisms

A disjoint union of directed cycles admits a conservative commutative binary (ccb) polymorphism (which coincide with conservative 2-semilattice operations) if and only if it contains no even cycles. This provides a case where Theorem 2 characterises a proper subclass of conservative Maltsev digraphs. In this section we classify Maltsev digraphs admitting a ccb polymorphism. A corollary of the result will be a sort of converse to Theorem 2: a Maltsev digraph with a ccb polymorphism is necessarily conservative Maltsev (Proposition 2 below).

Consider any digraph G and let $*$ be any conservative commutative binary operation on V_G. The operation $*$ has an easy interpretation as a colouring of the nondiagonal elements of the cartesian square $V_G^2 \backslash \{(v, v) \mid v \in V_G\}$: the pair (a, b) is coloured L if $a * b = a$ and R if $a * b = b$; commutativity is equivalent to (a, b) having different colour to (b, a). (The exclusion of the diagonal elements is only for convenience.) We now examine the consequences of $*$ being a ccb polymorphism.

For any digraph G we define a structure—the *ccb graph* of G—on the non-diagonal elements of the cartesian square $V_G^2 \backslash \{(v,v) \mid v \in V_G\}$. The ccb graph is a graph with two kinds of edges: "orienting" edges, which are directed, and "straight" edges which are considered as having no direction.

– A (directed) orienting edge is placed from (a_1,b_1) to (a_2,b_2) if there are parallel edges connecting each of the following pairs in G: a_1 and a_2; b_1 and b_2; and b_1 and a_2 *but not* a_1 and b_2. The following diagram depicts two situations that an orienting edge pointing from (a_1,b_1) to (a_2,b_2) can arise:



– An (undirected) straight edge is placed from (a_1,b_1) to (a_2,b_2) if there are parallel edges connecting the following pairs: a_1 and a_2; b_1 and b_2; but not a_1 and b_2 or a_2 and b_1.

A directed path in the ccb graph is a path of orienting edges and straight edges, in which each orienting edge is traversed in a forward direction. We now adopt the notation $\rightsquigarrow$ to denote directed paths: note that $(a,b) \rightsquigarrow (c,d)$ if and only if $(d,c) \rightsquigarrow (b,a)$. An important observation is: if (a,b) is not connected to (c,d) in the ccb graph of G then the compatibility of the ccb operation with the edges of G does not fail at the pairs (a,b) and (c,d). Thus *the digraph G admits a ccb polymorphism iff the ccb graph of G can be coloured by L and R such that L and R are preserved across straight edges, L is preserved forward across orienting edges and R is preserved backward across orienting edges.* Expressed in terms of $\rightsquigarrow$ this becomes the following rules.

(L) If (u,v) is coloured L and $(u,v) \rightsquigarrow (x,y)$ then:
  (1) (x,y) is coloured L.
  (2) (y,x) is coloured R.
(R) If (u,v) is coloured R and $(x,y) \rightsquigarrow (u,v)$ then:
  (1) (x,y) is coloured R.
  (2) (y,x) is coloured L.

Theorem 3. *A digraph G admits a ccb polymorphism $*$ if and only if for every distinct $a,b \in G$, the ccb-graph of G does not contain a directed path both from (a,b) to (b,a) and from (b,a) to (a,b). When a ccb polymorphism exists, it can be constructed in a polynomial number of steps.*

Proof. The forward implication is simply the statement that the colouring of the ccb-graph must colour each pair (a,b) oppositely to its reverse (b,a), and the colouring rules (L)(1)–(R)(2) must be obeyed.

Now we show the converse: assume that for every $a,b \in G$, the ccb-graph of G does not contain a directed path from (a,b) to (b,a) and from (b,a) to (a,b). We construct (in a polynomial number of steps) a successful colouring by L and R, whence a ccb polymorphism.

First phase. Begin by finding any pairs (a, b) from which there is a directed path to the reverse pair (b, a) (obviously, such a path must contain an orienting edge). In every case, colour (a, b) by R and (b, a) by L. By assumption, no pair is coloured two different colours simultaneously.

Second phase. Routine arguments show that after phase 1, no uncoloured pair is forced to be coloured by rules (L)(1)–(R)(2). Complete the colouring by iterating the following process until all pairs are coloured: take any uncoloured pair, (a, b) say, and colour it arbitrarily; colour any other pairs for which rules (L)(1)–(R)(2) apply starting from (a, b).

Routine arguments show that, at each iteration, no pair is coloured twice by different colours, hence the desired colouring (and ccb-polymorphism) is eventually achieved.

Note that the second property in Theorem 3 can be verified using Reachability in the ccb-graph of G, so is solvable in nondeterministic logarithmic space. Hence deciding if a digraph has a ccb polymorphism is in NL too.

Proposition 2. *The following are equivalent for a Maltsev graph G:*

1. *G has a conservative commutative idempotent binary polymorphism;*
2. *G has a conservative commutative idempotent binary polymorphism and a conservative Maltsev polymorphism;*
3. *G has a conservative Maltsev polymorphism and G_∞ is empty or has no even length cycles.*

Proof. (2)⇒(3)⇒(1) follow from Theorem 2, because a disjoint union of directed cycles has a ccb iff it has no even length cycles. For (1)⇒(2) we use Theorem 3 to prove a version of Lemma 7 with conservative 2-rectangularity replaced by ccb. The routine proof, which we omit, uses the digraph in Fig. 4.

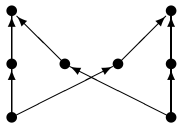


Fig. 4. A Maltsev digraph with no conservative Maltsev polymorphism

Note that the digraph in Fig. 4 is easily seen to admit a conservative majority polymorphism, a Maltsev polymorphism, but no conservative Maltsev polymorphism. So conservative majority cannot replace ccb in Proposition 2.

4.3 A Simple Inductive Construction of Maltsev DAGs

In this section we provide a simple inductive characterisation of totally rectangular DAGs. We note that in [19, Corollary 16] Kazda gives an inductive construction of Maltsev digraphs, however, this construction is not fully satisfying in the sense that it is non-deterministic, i.e. it does not specify how to obtain the desired preimages, and it is not clear if it can be made deterministic. The construction described below consists of repeated applications of two straightforward steps (and their reverse versions) which clearly specify how to obtain a new Maltsev digraph from an already constructed one by a certain copying process. We need the following definitions.

Definition 5 ((Reverse) arborescence). *An* (reverse) arborescence *is a directed tree with root r such that every edge points away from (towards) r.*

Definition 6 ($\nabla(r,h)$ and $\Delta(r,h)$). *Let G be a digraph, $r \in V_G$, and $h \in \mathbb{N}$. $\nabla(r,h)$ ($\Delta(r,h)$) is defined to be the subgraph of G whose vertices and edges are the vertices and edges of all (reverse) sub-dipaths of G which have initial vertex r and length h. A vertex $v \in \nabla(r,h) - r$ ($\Delta(r,h) - r$) is called an* endpoint *of $\nabla(r,h)$ ($\Delta(r,h)$) if there is a (reverse) dipath of length h from r to v. Otherwise v is called an* inner vertex *of $\nabla(r,h)$ ($\Delta(r,h)$).*

Definition 7 (isolated $\nabla(r,h)$). *Let G be a digraph. Consider $\nabla(r,h)$ ($\Delta(r,h)$) for some $r \in V_G$ and $h \in \mathbb{N}$. We say that $\nabla(r,h)$ ($\Delta(r,h)$) is* isolated *in G if for every inner vertex v of $\nabla(r,h)$, both the in-neighbourhood and the out-neighbourhood of v belongs to $\nabla(r,h)$ ($\Delta(r,h)$).*

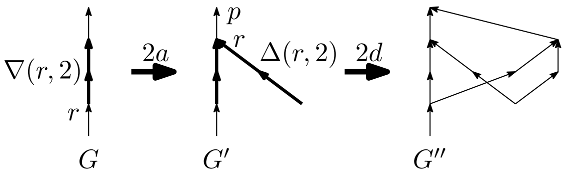


Fig. 5. Construction of a totally rectangular DAG

We are ready to define the construction formally in Fig. 6. This construction can be used, for example, to define a minority operation for a totally rectangular DAG. Also, a more restricted version of the construction can be defined, but it becomes slightly more technical.

Example 3. Consider the totally rectangular DAG G'' in Fig. 5. To construct it using the method in Fig. 6, we start with the dipath G and first apply Step 2a to G to obtain G'. Next we apply Step 2d to G' obtain G''. The thick edges indicate the subgraphs $\nabla(r,2)$ and $\Delta(r,2)$, which are the subgraphs to be copied and attached appropriately.

Theorem 4. *The class of totally rectangular DAGs is the set of digraphs $\mathcal{M}$ defined in Fig. 6.*

1. $\mathcal{C}$ contains dipaths of all possible lengths $n \in \mathbb{N}_0$;
2. $\mathcal{C}$ is closed under applying the following operations:
   (a) Given a digraph G, let $r \in V_G$ and $h \in \mathbb{N}$ such that $\nabla(r, h)$ is an arborescence. Let ∇' be a copy of $\nabla(r, h)$. Join ∇' to G by identifying the corresponding endpoints of ∇' and $\nabla(r, h)$. Let the resulting graph be G';
   (b) Given a digraph G, let $r \in V_G$ be such that r has exactly one incoming edge (p, r), and $h \in \mathbb{N}$ such that $\nabla(r, h)$ is an isolated arborescence. Let ∇' be a copy of $\nabla(r, h)$ with root r'. Join ∇' to C by identifying the corresponding endpoints of ∇' and $\nabla(r, h)$, and adding the edge (p, r'). Let the resulting graph be G';
   (c) The reverse version of Step 2a (defined in the natural way);
   (d) The reverse version of Step 2b (defined in the natural way).
3. $\mathcal{M}$ is the set of digraphs that can be obtained by taking disjoint unions of digraphs in $\mathcal{C}$.

Fig. 6. Inductive construction of the set $\mathcal{M}$ of totally rectangular DAGs

5 Some Applications to the Constraint Satisfaction Problem

The logic programming language Datalog is one of the main tools to solve CSPs in P. The fragments of Datalog called linear and symmetric Datalog are conjectured to contain all CSPs in NL and L, respectively, see [7,8,14]. A minor technicality is that it is actually the complement of a CSP that can be defined in Datalog and its fragments, not the actual CSP.

By Theorem 1, the core of a Maltsev digraph is either a directed path or a disjoint union of cycles, and for such digraphs the following is not difficult to show.

Corollary 2. *Let H be a Maltsev digraph. Then the complement of* $\mathrm{CSP}(H)$ *can be defined in symmetric Datalog of constant width and therefore* $\mathrm{CSP}(H)$ *is in* L.

The *list homomorphism problem* for a digraph H, $\mathsf{LHOM}(H)$, is the following decision problem. Given an input digraph G and for each vertex $v \in V_G$ a list $L_v \subseteq V_H$, determine if there is a homomorphism h from G to H such that for each $v \in V_G$, $h(v) \in L_v$. This problem is exactly $\mathrm{CSP}(H_u)$ where H_u is the structure obtained by expanding the digraph H with unary relations U, where U runs through all non-empty subsets of V_H . Using Corollary 1, the following corollary is easy to deduce.

Corollary 3. *The complement of* $\mathsf{LHOM}(H)$ *for a conservative Maltsev digraph H can be defined in symmetric Datalog, and therefore* $\mathsf{LHOM}(H)$ *is in* L.

References

1. Barto, L., Bulin, J.: CSP dichotomy for special polyads (2011) (submitted)
2. Barto, L., Kozik, M., Maróti, M., Niven, T.: CSP dichotomy for special triads. Proceedings of the AMS 137, 2921–2934 (2009)
3. Barto, L., Kozik, M., Niven, T.: Graphs, polymorphisms and the complexity of homomorphism problems. In: Proceedings of the 40th annual ACM symposium on Theory of computing, STOC 2008, pp. 789–796 (2008)
4. Bulatov, A., Dalmau, V.: A simple algorithm for Mal'tsev constraints. SIAM Journal on Computing 36(1), 16–27 (2006)
5. Bulatov, A., Jeavons, P., Krokhin, A.: Constraint satisfaction problems and finite algebras. In: Proceedings of the 27th International Colloquium on Automata, Languages and Programming, ICALP 2000, pp. 272–282 (2000)
6. Bulatov, A., Jeavons, P., Krokhin, A.: Classifying the complexity of constraints using finite algebras. SIAM Journal on Computing 34(3), 720–742 (2005)
7. Bulatov, A.A., Krokhin, A.A., Larose, B.: Dualities for constraint satisfaction problems. In: Creignou, N., Kolaitis, P.G., Vollmer, H. (eds.) Complexity of Constraints. LNCS, vol. 5250, pp. 93–124. Springer, Heidelberg (2008)
8. Dalmau, V.: Linear Datalog and bounded path duality of relational structures. Logical Methods in Computer Science 1, 1–32 (2005)
9. Dalmau, V., Krokhin, A.: Majority constraints have bounded pathwidth duality. European Journal of Combinatorics 29(4), 821–837 (2008)
10. Dalmau, V., Larose, B.: Maltsev + Datalog $\Rightarrow$ Symmetric Datalog. In: Proceedings of the 23rd IEEE Symposium on Logic in Computer Science, LICS 2008, pp. 297–306 (2008)
11. Dyer, M.E., Richerby, D.: On the complexity of #CSP. In: Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, pp. 725–734 (2010)
12. Egri, L.: On CSPs below P and the NL $\neq$ P conjecture (2011) (submitted)
13. Egri, L., Krokhin, A.A., Larose, B., Tesson, P.: The complexity of the list homomorphism problem for graphs. In: Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science, STACS 2010, pp. 335–346 (2010)
14. Egri, L., Larose, B., Tesson, P.: Symmetric Datalog and constraint satisfaction problems in logspace. In: Proceedings of the 22nd Annual IEEE Symposium on Logic in Computer Science, LICS 2007, pp. 193–202 (2007)
15. Hagemann, J., Mitschke, A.: On n-permutable congruences. Algebra Universalis 3, 8–12 (1973)
16. Hell, P., Nešetřil, J.: On the complexity of H-coloring. Journal of Combinatorial Theory, Series B 48, 92–110 (1990)
17. Hell, P., Rafiey, A.: The dichotomy of list homomorphisms for digraphs. In: Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, pp. 1703–1713 (2011)
18. Jeavons, P., Cohen, D., Gyssens, M.: Closure properties of constraints. J. ACM 44(4), 527–548 (1997)
19. Kazda, A.: Maltsev digraphs have a majority polymorphism. European Journal of Combinatorics 32, 390–397 (2011)
20. Maróti, M., McKenzie, R.: Existence theorems for weakly symmetric operations. Algebra Universalis 59(3-4), 463–489 (2008)

Join-Reachability Problems in Directed Graphs[⋆]

Loukas Georgiadis[1], Stavros D. Nikolopoulos[2], and Leonidas Palios[2]

[1] Department of Informatics and Telecommunications Engineering, University of Western Macedonia, Greece
`lgeorg@uowm.gr`

[2] Department of Computer Science, University of Ioannina, Greece
`{stavros,palios}@cs.uoi.gr`

Abstract. For a given collection $\mathcal{G}$ of directed graphs we define the *join-reachability graph* of $\mathcal{G}$, denoted by $\mathcal{J}(\mathcal{G})$, as the directed graph that, for any pair of vertices a and b, contains a path from a to b if and only if such a path exists in all graphs of $\mathcal{G}$. Our goal is to compute an efficient representation of $\mathcal{J}(\mathcal{G})$. In particular, we consider two versions of this problem. In the *explicit* version we wish to construct the smallest join-reachability graph for $\mathcal{G}$. In the *implicit* version we wish to build an efficient data structure (in terms of space and query time) such that we can report fast the set of vertices that reach a query vertex in all graphs of $\mathcal{G}$. This problem is related to the well-studied *reachability problem* and is motivated by emerging applications of graph-structured databases and graph algorithms. We consider the construction of join-reachability structures for two graphs and develop techniques that can be applied to both the explicit and the implicit problem. First we present optimal and near-optimal structures for paths and trees. Then, based on these results, we provide efficient structures for planar graphs and general directed graphs.

1 Introduction

In the *reachability problem* our goal is to preprocess a (directed or undirected) graph G into a data structure that can quickly answer queries that ask if a vertex b is reachable from a vertex a. This problem has numerous and diverse applications, including internet routing, geographical navigation, and knowledge-representation systems [13]. Recently, the interest in graph reachability problems has been rekindled by emerging applications of graph data structures in areas such as the semantic web, bio-informatics and social networks. These developments together with recent applications in graph algorithms [4,5,7] have motivated us to introduce the study of the *join-reachability problem* that we define as follows. We are given a collection $\mathcal{G}$ of λ directed graphs $G_i = (V_i, A_i)$, $1 \leq i \leq \lambda$, where each graph G_i represents a binary relation R_i over a set of elements $V \subseteq V_i$ in the following sense: For any $a, b \in V$, we have aR_ib if and only if b is reachable

[⋆] This research project has been funded by the John S. Latsis Public Benefit Foundation. The sole responsibility for the content of this paper lies with its authors.

from a in G_i. Let $\mathcal{R} \equiv \mathcal{R}(\mathcal{G})$ be the binary relation over V defined by: $a\mathcal{R}b$ if and only if aR_ib for all $i \in \{1, \ldots, \lambda\}$ (i.e., b is reachable from a in all graphs in $\mathcal{G}$). We can view $\mathcal{R}$ as a type of JOIN operation on graph-structured databases. Our objective is to find an efficient representation of this relation. To the best of our knowledge, this problem has not been previously studied. We will restrict our attention to the case of two input graphs ($\lambda = 2$).

Contribution. In this paper we explore two versions of the join-reachability problem. In the *explicit* version we wish to represent $\mathcal{R}$ with a directed graph $\mathcal{J} \equiv \mathcal{J}(\mathcal{G})$, which we call the *join-reachability graph of* $\mathcal{G}$, i.e., for any $a, b \in V$, we have $a\mathcal{R}b$ if and only if b is reachable from a in $\mathcal{J}$. Our goal is to minimize the size (i.e., the number of vertices plus arcs) of $\mathcal{J}$. We consider this problem in Sections 2 and 3, and present results on the computational and combinatorial complexity of $\mathcal{J}$. In the *implicit* version we wish to represent $\mathcal{R}$ with an efficient data structure (in terms of space and query time) that can report fast all elements $a \in V$ satisfying $a\mathcal{R}b$ for any query element $b \in V$. We deal with the implicit problem in Section 4. First we describe efficient join-reachability structures for simple graph classes. Then, based on these results, we consider planar graphs and general directed graphs. Although we focus on the case of two directed graphs ($\lambda = 2$), we note that some of our results are easily extended for $\lambda \geq 3$ with the use of appropriate multidimensional geometric structures.

Applications. Instances of the join-reachability problem appear in various applications. For example, in the rank aggregation problem [3] we are given a collection of rankings of some elements and we may wish to report which (or how many) elements have the same ranking relative to a given element. This is a special version of join-reachability since the given collection of rankings can be represented by a collection of directed paths with the elements being the vertices of the paths. Similarly, in a graph-structured database with an associated ranking of its vertices we may wish to find the vertices that are related to a query vertex and have higher or lower ranking than this vertex. Instances of join-reachability also appear in graph algorithms arising from program optimization. Specifically, [4] uses a data structure that reports fast vertices that satisfy certain ancestor-descendant relations in a collection of rooted trees. Moreover, in [7] it is shown that any directed graph G with a distinguished source vertex s has two spanning trees rooted at s such that a vertex a is a dominator of a vertex b (meaning that all paths in G from s to b pass through a) if and only if a is an ancestor of b in both spanning trees. This generalizes the graph-theoretical concept of *independent spanning trees*. Two spanning trees of a graph G are independent if they are both rooted at the same vertex r and for each vertex v the paths from r to v in the two trees are internally vertex disjoint. Similarly, λ spanning trees of G are independent if they are pairwise independent. In this setting, we can apply a join-reachability structure to decide if λ given spanning trees are independent. Finally we note that a variant of the join-reachability problem we defined here appears in the context of a recent algorithm for computing two internally vertex-disjoint paths for any pair of query vertices in a 2-vertex connected directed graph [5].

Preliminaries and Related Work. The reachability problem is easy in the undirected case since it suffices to compute the connected components of the input graph. Similarly, the undirected version of the join-reachability problem is also easy, as given the connected components of two undirected graphs G_1 and G_2 with n vertices, we can compute the connected components of $\mathcal{J}(\{G_1, G_2\})$ in $O(n)$ time. On the other hand, no reachability data structure is currently known to simultaneously achieve $o(n^2)$ space and $o(n)$ query time for a general directed graph with n vertices [13]. Nevertheless, efficient reachability structures do exist for several important cases. First, asymptotically optimal structures exist for rooted trees [1] and planar directed graphs with one source and one sink [8,11]. For general planar graphs Thorup [12] gives an $O(n \log n)$-space structure with constant query time. Talamo and Vocca [10] achieve constant query time for lattice partial orders with an $O(n\sqrt{n})$-space structure.

Notation. In the description of our results we use the following notation and terminology. We denote the vertex set and the arc set of a directed graph (digraph) G by $V(G)$ and $A(G)$, respectively. Without loss of generality we assume that $V(G) = V$ for all $G \in \mathcal{G}$. The size of G, denoted by $|G|$, is equal to the number of arcs plus vertices, i.e., $|G| = |V| + |A|$. We use the notation $a \rightsquigarrow_G b$ to denote that b is reachable from a in G. (By definition $a \rightsquigarrow_G a$ for any $a \in V$.) The *predecessors* of a vertex b are the vertices that reach b, and the *successors* of a vertex b are the vertices that are reached from b. Let P be a directed path (dipath); the *rank* of $a \in P$, $r_P(a)$, is equal to the number of predecessors of a in P minus one, and the *height* of $a \in P$, $h_P(a)$, is equal to the number of successors of a in P minus one. For a rooted tree T, we let $T(a)$ denote the subtree rooted at a. We will deal with two special types of directed rooted trees: In an *in-tree*, each vertex has exactly one outgoing arc except for the root which has none; in an *out-tree*, each vertex has exactly one incoming arc except for the root which has none. We use the term *unoriented tree* for a directed tree with no restriction on the orientation of its arcs. Similarly, we use the term *unoriented dipath* to refer to a path in the undirected sense, where the arcs can have any orientation. In our constructions we map the vertices of V to objects in a d-dimensional space and use the notation $x_i(a)$ to refer to the ith coordinate that vertex a receives. Finally, for any two vectors $\xi = (\xi_1, \dots, \xi_d)$ and $\zeta = (\zeta_1, \dots, \zeta_d)$, the notation $\xi \le \zeta$ means that $\xi_i \le \zeta_i$ for $i = 1, \dots, d$.

1.1 Preprocessing: Computing Layers and Removing Cycles

Thorup's Layer Decomposition. In [12] Thorup shows how to reduce the reachability problem for any digraph G to reachability in some digraphs with special properties, called *2-layered digraphs*. A *t-layered spanning tree* T of G is a rooted directed tree such that any path in T from the root (ignoring arc directions) is the concatenation of at most t dipaths in G. A digraph G is *t-layered* if it has such a spanning tree. Now we provide an overview of Thorup's reduction. The vertices of G are partitioned into layers $L_0, L_1, \dots, L_{\mu-1}$ that define a sequence of digraphs $G^0, G^1, \dots, G^{\mu-1}$ as follows. An arbitrary vertex $v_0 \in V(G)$ is chosen as a root. Then, layer L_0 contains v_0 and the vertices that are reachable from v_0.

For odd i, layer L_i contains the vertices that reach the previous layers L_j, $j < i$. For even i, layer L_i contains the vertices that are reachable from the previous layers L_j, $j < i$. To form G^i for $i > 0$ we contract the vertices in layers L_j for $j \leq i-1$ to a single root vertex r_0; for $i = 0$ we set $r_0 = v_0$. Then G^i is induced by L_i, L_{i+1} and r_0. It follows that each G^i is a 2-layered digraph. Let $\iota(v)$ denote the index of the layer containing v, that is, $\iota(v) = i$ if and only if $v \in L_i$. The key properties of the decomposition are: (i) all the predecessors of v in G are contained in $G^{\iota(v)-1}$ and $G^{\iota(v)}$, and (ii) $\sum_i |G^i| = O(|G|)$.

Removing Cycles. In the standard reachability problem, a useful preprocessing step that can reduce the size of the input digraph is to contract its strongly connected components (strong components) and consider the resulting acyclic graph. When we apply the same idea to join-reachability we have to deal with the complication that the strong components in the two digraphs may differ. Still, we can construct two acyclic digraphs $\hat{G}_1$ and $\hat{G}_2$ such that, for any $a, b \in V$, $a \leadsto_{\mathcal{J}(G_1,G_2)} b$ if and only if $a \leadsto_{\mathcal{J}(\hat{G}_1,\hat{G}_2)} b$, and $|\hat{G}_i| \leq |G_i|$, $i = 1, 2$. This is accomplished as follows. First, we compute the strong components of G_1 and G_2 and order them topologically. Let G'_i, $i = 1, 2$, denote the digraph produced after contracting the strong components of G_i. (We remove loops and duplicate arcs so that each G'_i is a simple digraph.) Also, let C_i^j denote the jth strong component of G_i. We partition each component C_i^j into subcomponents such that two vertices are in the same subcomponent if and only if they are in the same strong component in both G_1 and G_2. The subcomponents are the vertices of $\hat{G}_1$ and $\hat{G}_2$. Next we describe how to add the appropriate arcs. The process is similar for the two digraphs so we consider only $\hat{G}_1$.

Let $C_1^{j,1}, C_1^{j,2}, \ldots, C_1^{j,l_j}$ be the subcomponents of C_1^j, which are ordered with respect to the topological order of G'_2. That is, if $x \in C_1^{j,i}$ and $y \in C_1^{j,i'}$, where $i < i'$, then in the topological order of G'_2 the component of x precedes the component of y. We connect the subcomponents by adding the arcs $(C_1^{j,i}, C_1^{j,i+1})$ for $1 \leq i < l_j$. Moreover, for each arc (C_1^i, C_1^j) in $A(G'_1)$ we add the arc $(C_1^{i,l_i}, C_1^{j,1})$ to $A(\hat{G}_1)$, where C_1^{i,l_i} is the last subcomponent of C_1^i. It is straightforward to verify that $a \leadsto_{\mathcal{J}} b$ if and only if a and b are in the same subcomponent or the subcomponent of a is a predecessor of the subcomponent of b in both $\hat{G}_1$ and $\hat{G}_2$.

2 Computational Complexity

We explore the computational complexity of computing the smallest $\mathcal{J}(\{G_1, G_2\})$: Given two digraphs $G_1 = (V, A_1)$ and $G_2 = (V, A_2)$ we wish to compute a digraph $\mathcal{J} \equiv \mathcal{J}(\{G_1, G_2\})$ of minimum size such that for any $a, b \in V$, $a \leadsto_{\mathcal{J}} b$ if and only if $a \leadsto_{G_1} b$ and $a \leadsto_{G_2} b$. We consider two versions of this problem, depending on whether $\mathcal{J}$ is allowed to have Steiner vertices (i.e., vertices not in V) or not: In the *unrestricted* version $V(\mathcal{J}) \supseteq V$, while in the *restricted* version $V(\mathcal{J}) = V$. Computing $\mathcal{J}$ is NP-hard in the unrestricted case. This is implied by a straightforward reduction from the *reachability substitute problem*, which was shown to be NP-hard by Katriel et al. [9]. In this problem we are given a digraph H and a subset $U \subseteq V(H)$, and ask for the smallest digraph H^* such that for any $a, b \in U$, $a \rightsquigarrow_{H^*} b$ if and only if $a \rightsquigarrow_H b$. For the reduction, we let $G_1 = H$ and let G_2 contain all the arcs connecting vertices in U only, that is, $A(G_2) = U \times U$. Clearly, for any $a, b \in U$ we have $a \rightsquigarrow_{\mathcal{J}} b$ if and only if $a \rightsquigarrow_H b$. Therefore computing the smallest join-reachability graph is equivalent to computing H^*. In the restricted case, on the other hand, we can compute $\mathcal{J}$ using transitive closure and transitive reduction computations, which can be done in polynomial time [2].

Theorem 1. *Let $\mathcal{J}$ be the smallest join-reachability graph of a collection of digraphs. The computation of $\mathcal{J}$ is feasible in polynomial time if Steiner vertices are not allowed, and NP-hard otherwise.*

Note that allowing Steiner vertices can reduce the size of $\mathcal{J}$ significantly. In Section 3 we explore the combinatorial complexity of the unrestricted join-reachability graph and provide bounds for $|\mathcal{J}|$ in several cases.

3 Combinatorial Complexity

In this section we provide bounds on the size of $\mathcal{J}(\{G_1, G_2\})$ for several types of graphs. These are summarized in the next theorem.

Theorem 2. *Given two digraphs G_1 and G_2 with n vertices, the following bounds on the size of the join-reachability graph $\mathcal{J}(\{G_1, G_2\})$ hold:*

(a) $\Theta(n \log n)$ in the worst case when G_1 is an unoriented tree and G_2 is an unoriented dipath.
(b) $O(n \log^2 n)$ when both G_1 and G_2 are unoriented trees.
(c) $O(n \log^2 n)$ when G_1 is a planar digraph and G_2 is an unoriented dipath.
(d) $O(n \log^3 n)$ when both G_1 and G_2 are planar digraphs.
(e) $O(\kappa_1 n \log n)$ when G_1 is a digraph that can be covered with κ_1 vertex-disjoint dipaths and G_2 is an unoriented dipath.
(f) $O(\kappa_1 n \log^2 n)$ when G_1 is a digraph that can be covered with κ_1 vertex-disjoint dipaths and G_2 is a planar graph.
(g) $O(\kappa_1 \kappa_2 n \log n)$ when each G_i, $i = 1, 2$, is a digraph that can be covered with κ_i vertex-disjoint dipaths.

In the following sections we prove Theorem 2. In each case we provide a construction of the corresponding join-reachability graph that achieves the claimed bound. In Section 4 we provide improved space bounds for the implicit representation of $\mathcal{J}(\{G_1, G_2\})$, i.e., data structures that answer join-reachability reporting queries fast. Still, a process that computes an explicit representation of $\mathcal{J}(\{G_1, G_2\})$ can be useful, as it provides a natural way to handle collections of more than two digraphs (i.e., it allows us to combine the digraphs one pair at a time).

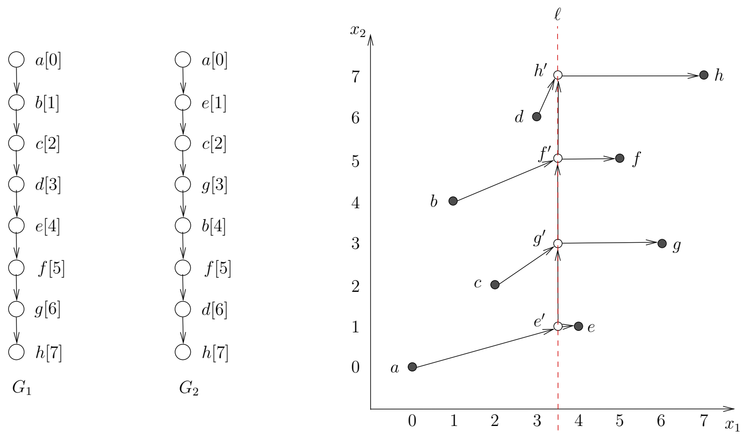


Fig. 1. The mapping of the vertices of two dipaths to a 2d rank space and the construction of $\mathcal{J}_\ell$; Steiner vertices in $\mathcal{J}_\ell$ are shown white

3.1 Two Paths

We start with the simplest case where G_1 and G_2 are dipaths with n vertices. First we show that we can construct a join-reachability graph of size $O(n \log n)$. Given this result we can provide bounds for trees, planar and general digraphs. Then we show this bound is tight, i.e., there are instances for which $\Omega(n \log n)$ size is needed. We begin by mapping the vertices of V to a two-dimensional rank space: Each vertex a receives coordinates $(x_1(a), x_2(a))$ where $x_1(a) = r_{G_1}(a)$ and $x_2(a) = r_{G_2}(a)$. Note that these ranks are integers in the range $[0, n-1]$. Now we can view these vertices as lying on an $n \times n$ grid, such that each row and each column of the grid contains exactly one vertex. Clearly, $a\mathcal{R}b$ if and only if $(x_1(a), x_2(a)) \leq (x_1(b), x_2(b))$.

Upper bound. We use a simple divide-and-conquer method. Let ℓ be the vertical line with x_1-coordinate equal to $n/2$. A vertex z is *to the right of* ℓ if $x_1(z) \geq n/2$ and *to the left of* ℓ otherwise. The first step is to construct a subgraph $\mathcal{J}_\ell$ of $\mathcal{J}$ that connects the vertices to the left of ℓ to the vertices to the right of ℓ. For each vertex b to the right of ℓ we create a Steiner vertex b' and add the arc (b', b). Also, we assign to b' the coordinates $(n/2, x_2(b))$. We connect these Steiner vertices in a dipath starting from the vertex with the lowest x_2-coordinate. Next, for each vertex a to the left of ℓ we locate the Steiner vertex b' with the smallest x_2-coordinate such that $x_2(a) \leq x_2(b')$. If b' exists we add the arc (a, b'). See Figure 1. Finally we recurse for the vertices to the left of ℓ and for the vertices to the right of ℓ. It is easy to see that $\mathcal{J}$ contains a path from a to b if and only if $(x_1(a), x_2(a)) \leq (x_1(b), x_2(b))$. To bound $|\mathcal{J}|$ note that we have $O(\log n)$ levels of recursion, and at each level the number of added Steiner vertices and arcs is $O(n)$. Hence, the $O(n \log n)$ bound for two dipaths follows.

The case of two unoriented dipaths G_1 and G_2 can be reduced to that of dipaths, yielding the same $O(n \log n)$ bound. This is accomplished by splitting

G_1 and G_2 to maximal subpaths that consist of arcs with the same orientation. Then $\mathcal{J}$ is formed from the union of separate join-reachability graphs for each pair of subpaths of G_1 and G_2. The $O(n \log n)$ bound follows from the fact that each vertex appears in at most two subpaths of each unoriented dipath, so in at most four subgraphs. We remark that our construction can be generalized to handle more dipaths, with an $O(\log n)$ factor blowup per additional dipath.

Lower bound. Let G_1 be any dipath, and let $x_1(a) = r_{G_1}(a)$. Also let $x_1^i(a)$ denote the ith bit in the binary representation of $x_1(a)$ and let $\beta = \lceil \log_2 n \rceil$ be the number of bits in this representation. We use similar notation for $x_2(a)$. We define G_2 such that the rank of a in G_2 is $x_2(a) = x_1(a)^R$, where $x_1(a)^R$ is the integer formed by the bit-reversal in the binary representation of $x_1(a)$, i.e., $x_2^i(a) = x_1^{\beta-1-i}(a)$ for $0 \le i \le \beta - 1$. Let $\mathcal{P}$ be the set that contains all pairs of vertices (a, b) that satisfy $x_1^i(a) = 0$, $x_1^i(b) = 1$ and $x_1^j(a) = x_1^j(b), j \neq i$, for $0 \le i \le \beta - 1$. Notice that for a pair $(a, b) \in \mathcal{P}$, $x_1(a) < x_1(b)$ and $x_1(a)^R < x_1(b)^R$. Hence $(x_1(a), x_2(a)) < (x_1(b), x_2(b))$, which implies $a \leadsto_{\mathcal{J}} b$. Now let G be the digraph that is formed by the arcs $(a, b) \in \mathcal{P}$. Then $a \leadsto_G b$ only if $a \leadsto_{\mathcal{J}} b$. Moreover, the transitive reduction of G is itself and has size $\Omega(n \log n)$. We also observe that any two vertices in G share at most one immediate successor. Therefore the size of G cannot be reduced by introducing Steiner vertices. This implies that size of $\mathcal{J}$ is also $\Omega(n \log n)$.

3.2 Tree and Path

Let G_1 be a rooted (in- or out-)tree and G_2 a dipath. First we note that the ancestor-descendant relations in a rooted tree can be described by two linear orders (corresponding to a preorder and a postorder traversal of the tree) and therefore we can get an $O(n \log^2 n)$ bound on the size of $\mathcal{J}$ using the result of Section 3.1. Here we provide an $O(n \log n)$ bound, which also holds when G_1 is unoriented. This upper bound together with the $\Omega(n \log n)$ lower bound of Section 3.1 implies Theorem 2(a).

Let T be the rooted tree that results from G_1 after removing arc directions. We associate each vertex $x \in T$ with a label $h(x) = h_{G_2}(x)$, the height of x in G_2. If G_1 is an out-tree then any vertex b must be reachable from all its ancestors a in T with $h(a) > h(b)$. Similarly, if G_1 is an in-tree then any vertex b must be reachable from all its descendants a in T with $h(a) > h(b)$. We begin by assigning a depth-first search interval to each vertex in T. Let $I(a) = [s(a), t(a)]$ be the interval of a vertex $a \in T$; $s(a)$ is the time of the first visit to a (during the depth-first search) and $t(a)$ is the time of the last visit to a. These times are computed by incrementing a counter after visiting or leaving a vertex during the search. This way all the $s()$ and $t()$ values that are assigned are distinct and for any vertex a we have $1 \le s(a) < t(a) \le 2n$. Moreover, by well-known properties of depth-first search, we have that a is an ancestor of b in T if and only if $I(b) \subseteq I(a)$; if a and b are unrelated in T then $I(a)$ and $I(b)$ do not intersect. Now we map each vertex a to the x_1-axis-parallel segment $S(a) = I(a) \times h(a)$.

As in Section 3.1 we use a divide-and-conquer method to build $\mathcal{J}$. We will consider G_1 to be an out-tree; the in-tree case is handled similarly and yields the same asymptotic bound. Let ℓ be the horizontal line with x_2-coordinate equal to $n/2$. A vertex x is *above* ℓ if $h(x) \geq n/2$; otherwise ($h(x) < n/2$), x is *below* ℓ. We create a subgraph $\mathcal{J}_\ell$ of $\mathcal{J}$ that connects the vertices above ℓ to the vertices below ℓ. To that end, for each vertex u above ℓ we create a Steiner vertex u' together with the arc (u, u'). Let z be the nearest ancestor of u in T that is above ℓ. If z exists then we add the arc (z', u'). Then, for each vertex y below ℓ we locate the nearest ancestor u of y in T that is above ℓ. If u exists then we add the arc (u', y). Finally, we recurse for the vertices above ℓ and for the vertices below ℓ.

It is not hard to verify the correctness of the above construction. The size of the resulting graph can be bounded by $O(n \log n)$ as in Section 3.1. Furthermore, we can generalize this construction for an unoriented tree and an unoriented path, and accomplish the same $O(n \log n)$ bound as required by Theorem 2(a). We omit the details which are similar to the more complicated construction of Section 3.4.

3.3 Two Trees

The construction of Section 3.2 can be extended to handle more than one dipath. We show how to apply this extension in order to get an $O(n \log^2 n)$ bound for the join-reachability graph of two rooted trees. We consider the case where G_1 is an out-tree and G_2 is an in-tree; the other two cases (two out-trees and two in-trees) are handled similarly.

Let T_1 and T_2 be the corresponding undirected trees. We assign to each vertex a two depth-first search intervals $I_1(a) = [s_1(a), t_1(a)]$ and $I_2(a) = [s_2(a), t_2(a)]$, where $I_j(a)$ corresponds to T_j, $j = 1, 2$. We create two linear orders (i.e., dipaths), P_1 and P_2, from the I_2-intervals as follows: In P_1 the vertices are ordered by decreasing s_2-value and in P_2 by increasing t_2-value. Each vertex a is mapped to an x_1-axis-parallel segment $I_1(a) \times x_2(a) \times x_3(a)$ (in three dimensions), where $x_2(a) = h_{P_1}(a)$ and $x_3(a) = h_{P_2}(a)$. Then b is reachable from a in $\mathcal{J}$ if and only if $I_1(b) \subseteq I_1(a)$ and $(x_2(b), x_3(b)) \leq (x_2(a), x_3(a))$. See Figure 2.

Again we employ a divide-and-conquer approach and use the method of Section 3.2 as a subroutine. The details are as follows. Let p be the plane with x_3-coordinate equal to $n/2$. We construct a subgraph $\mathcal{J}_p$ of $\mathcal{J}$ that connects the vertices above p (i.e., vertices z with $x_3(z) \geq n/2$) to the vertices below p (i.e., vertices z with $x_3(z) < n/2$). Then we use recursion for the vertices above p and the vertices below p.

We construct $\mathcal{J}_p$ using the method of Section 3.2 with some modifications. Let ℓ be the horizontal line with x_2-coordinate equal to $n/2$. We create a subgraph $\mathcal{J}_{p,\ell}$ of $\mathcal{J}_p$ that connects the vertices above p and ℓ to the vertices below p and ℓ. To that end, for each vertex z with $(x_2(z), x_3(z)) \geq (n/2, n/2)$ we create a Steiner vertex z' together with the arc (z, z'). Let u be the nearest ancestor of z in T_1 such that $(x_2(u), x_3(u)) \geq (n/2, n/2)$. If u exists then we add the arc (u', z'). Finally, for each vertex y with $(x_2(y), x_3(y)) < (n/2, n/2)$ we locate the nearest ancestor z of y in T_1 such that $(x_2(z), x_3(z)) \geq (n/2, n/2)$. If z exists then we add the arc (z', y). Finally, we recurse for the vertices above ℓ and for

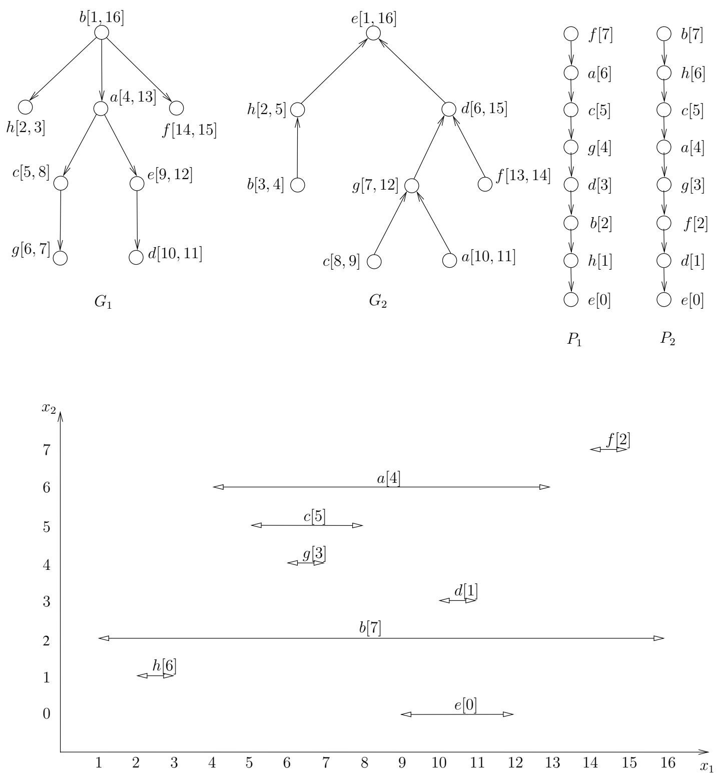


Fig. 2. The mapping of the vertices of two rooted trees to horizontal segments in 3d. The value in brackets above the segments correspond to the x_3-coordinates.

the vertices below ℓ. The above construction implies that $a \rightsquigarrow_{\mathcal{J}} b$ if and only if $I_1(b) \subseteq I_1(a)$ and $(x_2(b), x_3(b)) \leq (x_2(a), x_3(a))$, as required.

Now we bound the size of our construction. From Section 3.2 we have that the size of each substructure $\mathcal{J}_p$ is $O(n \log n)$. Since each vertex participates in $O(\log n)$ such substructures, the total size is bounded by $O(n \log^2 n)$.

3.4 Unoriented Trees

We can reduce the case of unoriented trees to that of rooted trees by applying Thorup's layer decomposition (see Section 1.1). We apply this decomposition to both G_1 and G_2. Let $G_i^0, G_i^2, \ldots, G_i^{\mu_i - 1}$ be the sequence of rooted trees produced from G_i, $i = 1, 2$, where each G_i^j is a 2-layered tree. See Figure 3. For even j, G_i^j consists of a *core* out-tree, formed by the arcs directed away from the root, and a collection of *fringe* in-trees. The situation is reversed for odd j, where the core tree is an in-tree and the fringe trees are out-trees. We call a vertex of the core tree a *core vertex*; we call a vertex of a fringe tree (excluding its root) a *fringe vertex*.

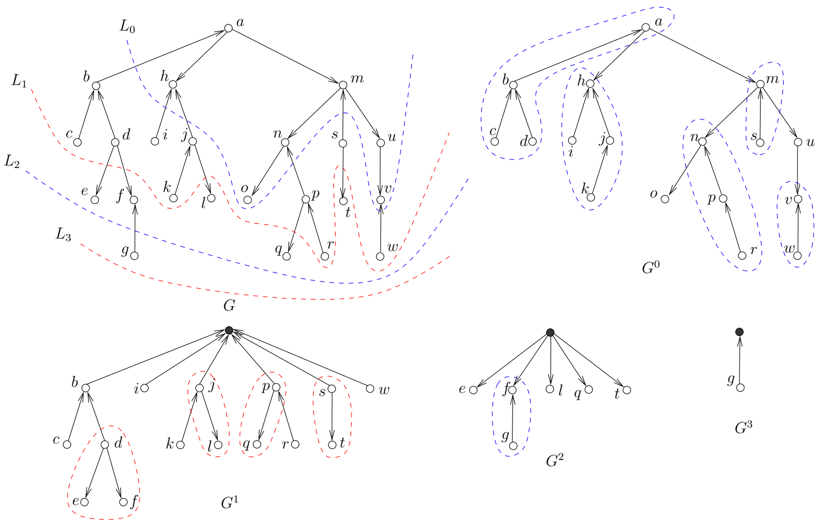


Fig. 3. An unoriented tree and its sequence of 2-layered trees. Fringe trees are encircled.

We build $\mathcal{J}$ as the union of join-reachability graphs $\mathcal{J}_{i,j}$ for each pair (G_1^i, G_2^j). Each graph $\mathcal{J}_{i,j}$ is constructed similarly to Section 3.3, with the exception that we have to take special care for the fringe vertices. (We also remark that in general $\mathcal{J}_{i,j} \neq \mathcal{J}(G_1^i, G_2^j)$.) A vertex $z \in V(G_1^i) \cap V(G_2^j)$ is included in $\mathcal{J}_{i,j}$ if one of the following cases hold: (i) z is a core vertex in at least one of G_1^i and G_2^j, or (ii) z is a fringe vertex in both G_1^i and G_2^j and the corresponding fringe trees containing z are either both in-trees or both out-trees. Let $V_{i,j}$ be the vertices in $V(G_1^i) \cap V(G_2^j)$ that satisfy the above condition.

If $V_{i,j} = \emptyset$ then $\mathcal{J}_{i,j}$ is empty. Now suppose $V_{i,j} \neq \emptyset$. First consider the case where the core of G_1^i is an out-tree. We contract each fringe in-tree to its root and let the new core supervertex correspond to the vertices of the contracted fringe tree. Let $\hat{G}_1^i$ be the out-tree produced from this process. Equivalently, if the core of G_1^i is an in-tree then the contraction of the fringe out-trees produces an in-tree $\hat{G}_1^i$. We repeat the same process for G_2^j. Next, we assign a depth-first search interval $I_1(z)$ to each vertex z in $\hat{G}_1^i$ and a depth-first search interval $I_2(z)$ to each vertex z in $\hat{G}_2^j$, as in Section 3.3. The vertices in $V_{i,j}$ are assigned a depth-first search interval in both trees, and therefore can be mapped to horizontal segments in a 3d space, as in Section 3.3. Hence, we can employ the method of Section 3.3 with some necessary changes that involve the fringe vertices. Let $z \in V_{i,j}$ be a fringe vertex in at least one of G_1^i and G_2^j. If the fringe tree containing z is an in-tree then we only include in $\mathcal{J}_{i,j}$ arcs leaving z; otherwise we only include arcs entering z.

Finally we need to show that the size of the resulting graph is $O(n \log^2 n)$. This follows from the fact that each subgraph $\mathcal{J}_{i,j}$ has size $O(n \log^2 n)$ and that each vertex can appear in at most four such subgraphs. Theorem 2(b) follows.

3.5 Planar Digraphs

Now we turn to planar digraphs and combine our previous constructions with Thorup's reachability oracle [12]. From this combination we derive the bounds stated in Theorem 2(c) and (d). First we need to provide some details for the reachability oracle of [12].

Let G be a planar digraph, and let $G^0, G^1, \dots, G^{\mu-1}$ be the sequence of 2-layered digraphs produced from G as described in Section 1.1. Consider one of these digraphs G^i. The next step is to obtain a separator decomposition of G^i. To that end, we treat G^i as an undirected graph and compute a separator S whose removal separates G^i into components, each with at most half the vertices. The separator S consists of three root paths of a spanning tree of G^i rooted at r_0. Because G^i is 2-layered, each root path in S corresponds to at most two dipaths in G^i. The key idea now is to process each separator dipath Q and find the connections between $V(G^i)$ and Q. For each $v \in V(G^i)$ two quantities are computed: (i) $\text{from}_v[Q]$ which is equal to $r_Q(u)$, where $u \in Q$ is the vertex with the highest rank in Q such that $u \rightsquigarrow_{G^i} v$, and (ii) $\text{to}_v[Q]$ which is equal to $r_Q(u)$, where $u \in Q$ is the vertex with the lowest rank in Q such that $v \rightsquigarrow_{G^i} u$. Clearly there is a path from a to b that passes though Q if and only if $\text{to}_a[Q] \leq \text{from}_b[Q]$. The same process is carried out recursively for each component of $G^i \setminus V(S)$. The depth of this recursion is $O(\log n)$, so each vertex is connected to $O(\log n)$ separator dipaths. The space and construction time for this structure is $O(n \log n)$.

Now we consider how to construct a join-reachability graph when G_1 is a planar digraph. We begin with the case where G_2 is a dipath. First we perform the layer decomposition of G_1 and construct the corresponding graph sequence $G_1^0, G_1^1, \dots, G_1^{\mu-1}$. Then we form pairs of digraphs $P_i = \{G_1^i, G_2^i\}$ where G_2^i is a dipath containing only the vertices in $V(G_1^i)$ in the order they appear in G_2. Clearly $a \rightsquigarrow_{\mathcal{J}} b$ if and only if $a \rightsquigarrow_{\mathcal{J}_{\iota(b)-1}} b$ or $a \rightsquigarrow_{\mathcal{J}_{\iota(b)}} b$, where $\mathcal{J}_i$ is the join-reachability graph of P_i. Then $\mathcal{J}$ is formed from the union of $\mathcal{J}_0, \dots, \mathcal{J}_{\mu-1}$.

To construct $\mathcal{J}_i$ we perform the separator decomposition of G_1^i, so that each vertex is associated with $O(\log n)$ separator dipaths. Let Q be such a separator dipath. Also, let V_Q be the set of vertices that have a successor or a predecessor in Q. We build a subgraph $\mathcal{J}_{i,Q}$ of $\mathcal{J}_i$ for the vertices in V_Q; $\mathcal{J}_i$ is formed from the union of the subgraphs $\mathcal{J}_{i,Q}$ for all the separator dipaths of G_1^i. The construction of $\mathcal{J}_{i,Q}$ is carried out as follows. Let $z \in V_Q$. If z has a predecessor in Q then we create a vertex z^- which is assigned coordinates $x_1(z^-) = \text{from}_z[Q]$ and $x_2(z^-) = r_{G_2}(z)$, and add the arc (z, z^-). Similarly, if z has a successor in Q then we create a vertex z^+ which is assigned coordinates $x_1(z^+) = \text{to}_z[Q]$ and $x_2(z^+) = r_{G_2}(z)$, and add the arc (z^+, z).

Now we can use the method of Section 3.1 to build the rest of $\mathcal{J}_{i,Q}$, so that $a \rightsquigarrow_{\mathcal{J}_{i,Q}} b$ if and only if $(x_1(a^+), x_2(a^+)) \leq (x_1(b^-), x_2(b^-))$. Let ℓ be the vertical line with x_1-coordinate equal to $n/2$. The first step is to construct the subgraph of $\mathcal{J}_{i,Q}$ that connects the vertices a^+ with $x_1(a^+) \leq n/2$ to the vertices b^- with $x_1(b^-) \geq n/2$. For each such b^- we create a Steiner vertex b' and add the arc (b', b^-). Also, we assign to b' the coordinates $(n/2, x_2(b^-))$. We connect these Steiner vertices in a dipath starting from the vertex with the lowest x_2-coordinate. Next, for each vertex a^+ with $x_1(a^+) \leq n/2$ we locate the Steiner vertex b' with the smallest x_2-coordinate such that $x_2(a^+) \leq x_2(b')$. If b' exists we add the arc (a^+, b'). Finally we recurse for the vertices with x_1-coordinate in $[1, n/2)$ and for the vertices with x_1-coordinate in $(n/2, n]$.

It remains to bound the size of $\mathcal{J}$. From Section 3.1, we have $|\mathcal{J}_{i,Q}| = O(|V_Q| \log |V_Q|)$. Moreover, the bound $\sum_Q |V_Q| = O(|V(G_1^i)| \log |V(G_1^i)|)$, where the sum is taken over all separator paths of G_1^i, implies $|\mathcal{J}_i| \leq \sum_Q |\mathcal{J}_{i,Q}| = O(|V(G_1^i)| \log^2 |V(G_1^i)|)$. Finally, since $\sum_i |V(G_1^i)| = O(n)$ we obtain $|\mathcal{J}| \leq \sum_i |\mathcal{J}_i| = O(n \log^2 n)$.

We handle the case where G_2 is an unordered dipath as noted in Section 3.1, which implies Theorem 2(c). The methods we developed here in combination with the structures of Section 3.4 result to a join-reachability graph of size $O(n \log^3 n)$ for a planar digraph and an unoriented tree. The same bound of $O(n \log^3 n)$ is achieved for two planar digraphs, as stated in Theorem 2(d).

3.6 General Digraphs

A technique that is used to speed up transitive closure and reachability computations is to cover a digraph with simple structures such as dipaths, chains, or trees (e.g., see [1]). Such techniques are well-suited to our framework as they can be combined with the structures we developed earlier. We also remark that the use of the preprocessing steps of Section 1.1 reduces the problem from general digraphs to acyclic and 2-layered digraphs. In this section we describe how to obtain join-reachability graphs with the use of dipath covers. This gives the bounds stated in Theorem 2(e)-(g); similar results can be derived with the use of tree covers. Again for simplicity, we first consider the case where G_1 is a general digraph and G_2 is a dipath.

A *dipath cover* is a decomposition of a digraph into vertex-disjoint dipaths. Let $P_1^1, P_1^2, \ldots P_1^{\kappa_1}$ be a dipath cover of G_1. For each vertex v and each path P_1^i we compute $\text{from}_v[P_1^i]$, i.e., $r_{P_1^i}(z)$ where $z \in P_1^i$ is the vertex with the highest rank in P_1^i such that $z \rightsquigarrow_{G_1} v$. Let P_2^i be the dipath that consists of the vertices in P_1^i ordered by increasing rank in G_2. Also, set $\text{from}_v[P_2^i] = r_{P_2^i}(z)$ where $z \in P_2^i$ is the vertex with the largest rank such that $r_{G_2}(z) \leq r_{G_2}(v)$. Let $V_{P_1^i}$ be set of vertices that have a predecessor in P_1^i. We build a subgraph $\mathcal{J}_i$ of $\mathcal{J}$ that connects the vertices of P_1^i to $V_{P_1^i}$. Then $\mathcal{J}$ is formed from the union of the subgraphs $\mathcal{J}_i$. For each $z \in V_{P_1^i}$ we create a vertex z^- which is assigned coordinates $x_1(z^-) = \text{from}_z[P_1^i]$ and $x_2(z^-) = \text{from}_z[P_2^i]$, and add the arc (z^-, z). Also, for each $z \in P_1^i$ we create a vertex z^+ which is assigned coordinates $x_1(z^+) = r_{P_1^i}(z)$ and $x_2(z^+) = r_{P_2^i}(z)$, and add the arc (z, z^+). Now we can build a join-reachability graph, so that $a \rightsquigarrow_{\mathcal{J}_i} b$ if and only if $(x_1(a^+), x_2(a^+)) \leq (x_1(b^-), x_2(b^-))$, as in Section 3.5.

The size of this graph is bounded by $\sum_i |V_{P_1^i}| \log |V_{P_1^i}| = O(\kappa_1 n \log n)$, which implies the result of Theorem 2(e). We can extend this method to handle two general digraphs and obtain the bound of Theorem 2(g). The case where G_2 is planar digraph is handled by combining the above method with the techniques of Section 3.5, resulting to Theorem 2(f).

4 Data Structures

Now we deal with the data structure version of the join-reachability problem. Our goal is to construct an efficient data structure for $\mathcal{J} = \mathcal{J}(G_1, G_2)$ such that given a query vertex b it can report all vertices a satisfying $a \leadsto_{\mathcal{J}} b$. We state the efficiency of a structure using the notation $\langle s(n), q(n,k) \rangle$ which refers to a data structure with $O(s(n))$ space and $O(q(n,k))$ query time for reporting k elements. In order to design efficient join-reachability data structures we apply the techniques we developed in Section 3. The bounds that we achieve this way are summarized in the following theorem. The proof is given in the full version of the paper [6].

Theorem 3. *Given two digraphs G_1 and G_2 with n vertices we can construct join-reachability data structures with the following efficiency:*

(a) $\langle n, k \rangle$ *when G_1 is an unoriented tree and G_2 is an unoriented dipath.*
(b) $\langle n, \log n + k \rangle$ *when G_1 is an out-tree and G_2 is an unoriented tree.*
(c) $\langle n \log^{\varepsilon} n, \log\log n + k \rangle$ *(for any constant $\varepsilon > 0$), when G_1 and G_2 are unoriented trees.*
(d) $\langle n \log n, k \log n \rangle$ *when G_1 is planar digraph and G_2 is an unoriented tree.*
(e) $\langle n \log^2 n, k \log^2 n \rangle$ *when both G_1 and G_2 are planar digraphs.*
(f) $\langle n\kappa_1, k \rangle$ *when G_1 is a general digraph that can be covered with κ_1 vertex-disjoint dipaths and G_2 is an unoriented tree.*
(g) $\langle n(\kappa_1 + \log n), k\kappa_1 \log n \rangle$ *or* $\langle n\kappa_1 \log n, k \log n \rangle$ *when G_1 is a general digraph that can be covered with κ_1 vertex-disjoint dipaths and G_2 is planar digraph.*
(h) $\langle n(\kappa_1 + \kappa_2), \kappa_1\kappa_2 + k \rangle$ *or* $\langle n\kappa_1\kappa_2, k \rangle$ *when each G_i, $i = 1, 2$, is a digraph that can be covered with κ_i vertex-disjoint dipaths.*

5 Conclusions and Open Problems

We considered the computational and combinatorial complexity of the join-reachability graph, and the design of efficient join-reachability data structures for a variety of graph classes. We believe that several open problems deserve further investigation. For instance, from the combinatorial complexity aspect, it would be interesting to prove or disprove that an $O(m \cdot \mathrm{polylog}(n))$ bound on the size of the join-reachability graph $\mathcal{J}(\{G_1, G_2\})$ is attainable when G_1 is a general digraph with n vertices and m arcs and G_2 is a dipath. Another direction is to consider the problem of approximating the smallest join-reachability graph for specific graph classes. From the data structures side, one can investigate how to support the following type of counting queries: Given a pair of query vertices compute the number of their common predecessors in $\mathcal{J}$. While some of our structures can be easily extended in order to support such counting queries, there are several cases (e.g., for planar digraphs) where we need to overcome various technical difficulties.

Acknowledgements. We would like to thank Li Zhang for several useful discussions.

References

1. Agrawal, R., Borgida, A., Jagadish, H.V.: Efficient management of transitive relationships in large data and knowledge bases. In: SIGMOD 1989: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, pp. 253–262 (1989)
2. Aho, A.V., Garey, M.R., Ullman, J.D.: The transitive reduction of a directed graph. SIAM J. Comput. 1(2), 131–137 (1972)
3. Dwork, C., Kumar, R., Naor, M., Sivakumar, D.: Rank aggregation methods for the web. In: WWW 2001: Proceedings of the 10th International Conference on World Wide Web, pp. 613–622 (2001)
4. Georgiadis, L.: Computing frequency dominators and related problems. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 704–715. Springer, Heidelberg (2008)
5. Georgiadis, L.: Testing 2-vertex connectivity and computing pairs of vertex-disjoint *s*-*t* paths in digraphs. In: Proc. 37th Int'l. Coll. on Automata, Languages, and Programming, pp. 738–749 (2010)
6. Georgiadis, L., Nikolopoulos, S.D., Palios, L.: Join-reachability problems in directed graphs. Technical Report arXiv:1012.4938v1 [cs.DS] (2010)
7. Georgiadis, L., Tarjan, R.E.: Dominator tree verification and vertex-disjoint paths. In: Proc. 16th ACM-SIAM Symp. on Discrete Algorithms, pp. 433–442 (2005)
8. Kameda, T.: On the vector representation of the reachability in planar directed graphs. Information Processing Letters 3(3), 75–77 (1975)
9. Katriel, I., Kutz, M., Skutella, M.: Reachability substitutes for planar digraphs. Technical Report MPI-I-2005-1-002, Max-Planck-Institut Für Informatik (2005)
10. Talamo, M., Vocca, P.: An efficient data structure for lattice operations. SIAM J. Comput. 28(5), 1783–1805 (1999)
11. Tamassia, R., Tollis, I.G.: Dynamic reachability in planar digraphs with one source and one sink. Theoretical Computer Science 119(2), 331–343 (1993)
12. Thorup, M.: Compact oracles for reachability and approximate distances in planar digraphs. Journal of the ACM 51(6), 993–1024 (2004)
13. Wang, H., He, H., Yang, J., Yu, P.S., Yu, J.X.: Dual labeling: Answering graph reachability queries in constant time. In: ICDE 2006: Proceedings of the 22nd International Conference on Data Engineering, p. 75 (2006)

Graphs of Bounded Treewidth Can Be Canonized in AC^1

Fabian Wagner

Inst. für Theoretische Informatik, Universität Ulm, Germany
fabian.wagner@uni-ulm.de

Abstract. In recent results the complexity of isomorphism testing on graphs of bounded treewidth is improved to TC^1 [17] and further to LogCFL [11]. The computation of canonical forms or a canonical labeling provides more information than isomorphism testing. Whether canonization is in NC or even TC^1 was stated as an open question in [18]. Köbler and Verbitsky [20] give a TC^2 canonical labeling algorithm. We show that a canonical labeling can be computed in AC^1. This is based on several ideas, e.g. that approximate tree decompositions of logarithmic depth can be obtained in logspace [15], and techniques of Lindells tree canonization algorithm [22]. We define recursively what we call a *minimal description* which gives with respect to some parameters in a logarithmic number of levels a canonical invariant together with an arrangement of all vertices. From this we compute a canonical labeling.

Keywords: Bounded Treewidth, Graph Isomorphism, Canonization.

1 Introduction

The *graph isomorphism problem* (GI) consists in deciding whether two given graphs are isomorphic, i.e. whether there is a permutation of all vertices that keeps the edge relation unchanged. GI is a well-studied problem in theoretical computer science because of its many applications and also, because it is one of the few natural problems in this class not known to be solvable in polynomial time, nor known to be NP-complete.

By studying GI, graph canonization receives a great attention because of its strong connection to GI. Thereby, it provides more information than isomorphism testing. Let $\mathcal{G}$ be a class of graphs. A *complete invariant* is a function $f : \mathcal{G} \to \{0,1\}^*$ where $f(G) = f(H)$ if and only if both graphs $G, H \in \mathcal{G}$ are isomorphic. With this function f, the isomorphism classes can be distinguished. A *canonical form* is a function $f : \mathcal{G} \to \mathcal{G}$ where $f(G)$ is a representative of the (equivalence) class of isomorphic graphs to G in $\mathcal{G}$. For example, if $f(G)$ is defined to be the lexicographically least graph in $\mathcal{G}$ isomorphic to G, then the computation of f is in general NP-hard (c.f. [3,23]). Clearly, graph isomorphism or the computation of a complete invariant is polynomial time reducible to graph canonization, the reverse direction is open. A *canonical labeling* assigns to each graph G in $\mathcal{G}$ a map σ that is an automorphism from G to the representative G^σ

of the class of isomorphic graphs to G in $\mathcal{G}$. Note, function f with $f(G) = G^\sigma$ is a canonical form. Canonical labeling is the strongest among these notions.

Robertson and Seymour [25] introduced the concept of bounded treewidth graphs, also known as *partial k-trees*. Intuitively speaking, the treewidth of a graph measures how much it differs from a tree. This concept has been used very successfully in algorithmics and fixed-parameter tractability, see for example [7,9]. Thereby, many problems that are NP-hard in general become efficiently solvable when restricted to graphs of bounded treewidth. Bodlaender showed in [5] that GI can be solved in polynomial time when restricted to this class of graphs. Grohe and Verbitsky [17] give a TC^1 upper bound, they use the Weisfeiler-Lehman algorithm that can be implemented as a logspace uniform family of TC^1-circuits. This bound was improved recently to LogCFL in [11].

For the canonization problems the situation is different. Köbler and Verbitsky [20] give an NC (in fact TC^2) canonical labeling algorithm. They prove a theorem showing that, for some special classes of graphs from a complete invariant that is computable in TC^k, a canonical labeling is computable in $\mathsf{TC}^{\mathsf{k}+1}$. With this theorem and with the fact that for bounded treewidth graphs a complete invariant can be computed in TC^1 [17], the result follows.

For many subclasses of bounded treewidth graphs, logspace algorithms are known, e.g. for canonization of trees [22], partial 2-trees [2], planar graphs [13], $K_{3,3}$-minor free and K_5-minor free graphs [14], and also for isomorphism testing on full k-trees [19].

Das, Torán, and Wagner [11] give an isomorphism algorithm that runs in LogCFL. The algorithm uses the fact that an arbitrary tree decomposition for one of the input graphs G can be computed in LogCFL [26] and by a recent result in L [15]. They set up the tree decomposition for the other graph H in parallel to the isomorphism test. For a graph there could be an exponential number of tree decompositions. That is one reason, why their algorithm cannot be generalized to a canonization algorithm.

The logspace-version of Courcelle's Theorem in [15] puts many problems on bounded treewidth graphs which can be formulated in *monadic second-order logic* in L, e.g. *3-colorability*, *Hamiltonicity*, or reachability problems. But it is not known how to formulate isomorphism testing or canonization problems in this logic. The research on the difficulty of computing tree decompositions of width at most k has a long history, there is a linear time bound of Bodlaender's Theorem [6] improving previous results [1,24]. Also the parallel time complexity has been reduced to $O(\log n)$ in a line of papers [10,4,21,8]. The space complexity has been reduced to LogCFL in [26] and recently to L by Elberfeld, Jakoby, and Tantau [15], where a logspace-version of Bodlaender's Theorem is shown: a tree decomposition for graphs of treewidth at most k can be computed in logspace. The following lemmas are important, to guarantee the treewidth bound of the input graph and a logarithmic depth bound.

Lemma 1. ([**15**]) *For every $k \geq 1$ the language* Tree-Width-k*, which contains exactly the graphs of treewidth at most k, is* L*-complete under first-order reductions.*

Lemma 2. ([15]) *Let G be a graph with n vertices and treewidth at most k. There is a tree decomposition that has width $4k + 3$ and depth at most $c \log_2 n$, where c is a constant depending only on k.*

The idea is, that for a split component of size m a child bag is defined in such a way that the split components of this child bag have size at most $m/2 + c$ with c a constant. This tree decomposition is called *approximate*.

Our contribution. The *existence* of a tree decomposition of logarithmic depth is the basis for our algorithm, to compute a canonical labeling in AC^1. The following observations are the key ingredients to guarantee this tight depth bound.

- Each bag is a separator, i.e. for each split component we have to find a child bag. Two child bags are connected via their parent bag only. Hence, the canonization process can be done for each child individually and in parallel.
- Many tasks are computed in preprocessing steps, e.g. whether the graph has tree width k, all possible bags of size $4k+3$, for each bag its possible children and the parent (with respect to a fixed root), or the split components for each possible bag and their size.
- The total number of possible bags of size $4k + 3$ in arbitrary tree decompositions is bounded by n^{4k+3}, where n is the size of the input graph. For each bag we define locally a circuit with unbounded fan-in gates. The total size of the circuit is also polynomial.
- We compute for each possible bag what we call a *minimal description*. It depends on some parameters, it consists of a unique description of the bag itself and a minimal description for the children, it is defined recursively.
- We limit the number of recursion levels of the minimal description to $O(\log n)$. Here we use Lemma 2, namely the *existence* of tree decompositions of logarithmic depth.
- A circuit of constant depth selects locally valid child bags to get the smallest minimal description. The difficult task is here sorting minimal descriptions. For this we use ideas from Lindells tree canonization algorithm [22]. Instead of a logarithmic space bound we have here logarithmic depth circuits.
- The length of the minimal description for a balanced subtree depends on its depth only. In the minimal description, a bag is described by its adjacency matrix in at most $(4k + 3)^2$ bits, if necessary we fill up to this total length repeating an extra symbol. We use the fact that a tree can be made *balanced* and also *binary* where its depth increases just by a constant factor (c.f. [15]).

The minimal description depends on the selection of the root bag, too. We run through all bags as roots and some further parameters in parallel. The minimal description so far is a *canonical invariant* for the input graph G. A *canonizing function* is obtained then as follows. While computing the minimal description we bring the vertices in a unique order. Then we use a simple logspace-computable procedure (see e.g. [12]) where vertices are renamed according to this order. This renaming is a *canonical labeling.* We get the main Theorem.

Theorem 1. *For every k, there is an AC^1-computable function that computes a canonical labeling for graphs of treewidth at most k.*

2 Preliminaries

We use the notion *interval* $[p, q]$ for the set $\{p, p+1, \ldots, q\}$. The *symmetric group* $Sym(V)$ is the set of all permutations of the elements in V.

A *word* or *string* is a tuple of symbols of an *alphabet* Σ. The *concatenation* of two words $W = w_1 w_2 \ldots w_k, W' = w'_1 w'_2 \ldots w'_k$ is denoted $WW' = w_1 w_2 \ldots w_k w'_1 w'_2 \ldots w'_k$.

Graphs. A *graph* G is a pair (V, E) with a set of *vertices* V (or $V(G)$) and *edges* $E \subseteq V \times V$ (or $E(G)$). If not stated otherwise, we consider simple graphs, i.e. without loops, directed edges or multi-edges. $G[X]$ is a subgraph of G *induced* on vertex set X, i.e. $G[X]$ has vertices X and edges $E(G[X]) = (X \times X) \cap E(G)$. Let $X \subseteq V$ then we write in short $G \setminus X = G[V(G) \setminus X]$. Let H be a subgraph of G, then $G \setminus H = G[V(G) \setminus V(H)]$.

A graph G is *connected* if there is a path between every pair of vertices in G. Let $U \subseteq V$ be a set of vertices. Let C be a connected component in $G \setminus U$ and let U' be those vertices from U connected to $V(C)$ in G. The induced subgraph of G on the set of vertices $V(C) \cup U'$ is a *split component* of U in G. We call U' the *minimal separating set of* C *in* U.

A *tree* is a connected graph that is free of cycles. Vertices in trees are also called *nodes*. A *root* of a tree is one designated node. A neighbor of a node is called *parent* if it is closer to the root and it is called *child* otherwise. A *leaf* of a tree has no children. In a *binary tree*, every node has at most two children. A binary tree with root is *balanced*, if for every node the number of nodes in its left and right subtrees differs by at most one. A binary tree is *perfect*, if it is balanced and every leaf is at the same depth. The *depth* of a tree is the longest distance from the root to a leaf. Let T and T' be trees rooted at r and r' and consider edges to be directed from roots to leafs. An *embedding of* T *into* T' is an injective mapping $\iota : V(T) \to V(T')$ where $\iota(r) = r'$ and for every pair $(a, b) \in V(T)$ there is a directed path from a to b iff there is a directed path from $\iota(a)$ to $\iota(b)$.

An *isomorphism* is a mapping ϕ of the vertices of one graph G onto the vertices of another graph H (we also write $G \cong H$) that preserves the edge relations, i.e. $\{u, v\} \in E(G) \Leftrightarrow \{\phi(u), \phi(v)\} \in E(H)$. Let $\mathcal{G}$ be a class of graphs. A *complete invariant* is a function $f : \mathcal{G} \to \{0, 1\}^*$ where $f(G) = f(H)$ iff $G \cong H$. A *canonical form* is a complete invariant with $f : \mathcal{G} \to \mathcal{G}$ where $f(G) \cong G$, we call $f(G)$ *canon* of G. A *canonical labeling* is a function $f : \mathcal{G} \to Sym(V(G))$ which assigns to a graph G an automorphism σ onto the canon, i.e. the function g with $g(G) = G^\sigma$ is a canonical form.

A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$, where $\{X_i \mid i \in I\}$ is a collection of subsets of V called *bags*, and T is a tree with node set I and edge set F, satisfying the following properties:

i) $\bigcup_{i \in I} X_i = V$
ii) for each $\{u, v\} \in E$, there is an $i \in I$ with $u, v \in X_i$ and
iii) for each $v \in V$, the set of nodes $\{i \mid v \in X_i\}$ forms a subtree of T.

The *width* of a tree decomposition $(\{X_i \mid i \in I\}, T = (I, F))$ of G is $\max\{|X_i| \mid i \in I\} - 1$. The *treewidth* of a graph G is the minimum width over all possible tree decompositions of G. An example is shown in Figure 1.

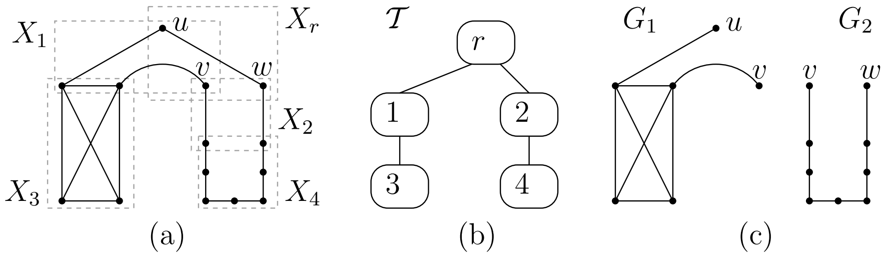


Fig. 1. (a) A graph G where dashed lines indicate bags $X_r, X_1, \ldots, X_4$.
(b) The set of bags form a tree decomposition $\mathcal{T}$ of G. Let r be the root.
(c) The split components G_1 and G_2 of X_r are shown. The sets $\{u, v\}, \{v, w\} \subseteq X_r$ are the minimal separating sets for G_1 and G_2, respectively.

Complexity. A *circuit* C_n is a finite directed acyclic graph with vertices associated to n input variables or gates (e.g. Boolean functions from a given base). For an assignment of the variables we associate a Boolean value to every gate in the circuit. The value of an input is the one given by the assignment to the corresponding variable. For an internal gate, the value is the one computed by the corresponding function, from the values of the gate inputs. The circuit computes a function $f(x_1, \ldots, x_n)$. This is the value of the designated *output gate*. The indegree of the vertices is called *fanin*. A *circuit family* $\{C_1, C_2, \ldots\}$ is a collection of circuits where C_n has n inputs. We consider here DLOGTIME-*uniform* circuit families, i.e. a deterministic Turing machine on input of 1^n, integer i, and bit b, with $O(\log n)$ time bound accepts iff the i-th bit of the description of C_n is b.

L (also denoted *logspace*) is the class of decision problems computable by deterministic logarithmic space Turing machines. LogCFL consists of all decision problems that can be Turing reduced in logspace to a context free language. Problems in LogCFL can be computed also by uniform families of polynomial size and logarithmic depth circuits over bounded fan-in *and*-gates and unbounded fan-in *or*-gates. The class NC^i contains the problems computable by uniform families of polynomial size and $O(\log^i n)$ depth circuits over bounded fan-in *and*-gates and bounded fan-in *or*-gates. Note, that $\mathsf{NC} = \bigcup_i \mathsf{NC}^i$. AC^i is defined as NC^i but with unbounded fan-in gates. The class TC^i contains the problems computable by uniform families of polynomial size and $O(\log^i n)$ depth circuits with threshold gates, i.e. gates that evaluate to 1 if at least half of their inputs are 1. The known relationships among these classes are: $\mathsf{AC}^0 \subset \mathsf{TC}^0 \subseteq \mathsf{NC}^1 \subseteq \mathsf{L} \subseteq \mathsf{LogCFL} \subseteq \mathsf{AC}^1 \subseteq \mathsf{TC}^1 \subseteq \mathsf{NC}^2 \subseteq \mathsf{AC}^2 \subseteq \mathsf{TC}^2 \subseteq \cdots \subseteq \mathsf{NC} \subseteq \mathsf{P} \subseteq \mathsf{NP}$.

3 Canonization of Graphs of Bounded Treewidth

To prove Theorem 1, we construct a circuit where we have some preprocessing steps, e.g. for the split components of bags and their size. Then in $O(\log n)$ steps, we compute for each bag X what we call a *good minimal description*, i.e. a unique tree decomposition that depends on some parameters. After the first level, we have good minimal descriptions for single bags, after the second level for bags which have leaf bags as children, and so on. After $O(\log n)$ levels, we have good minimal descriptions for tree decompositions of logarithmic depth, and by Lemma 2 a tree decomposition for the whole graph, if it exists. At each level, we select the smallest good minimal description. Once we find a good minimal description at some level i, then this remains unchanged in all levels $\geq i$. First, we describe some tools and then the construction of the minimal description.

3.1 Pre-ordering for Canonization and Valid Child Bags

Canonical Representation of Bags. Comparing adjacency matrices lexicographically is a natural way to test isomorphism. In general, this is an NP-complete problem, but when comparing bags this can be done with constant effort. We define $adj(G[X], \sigma)$ to be the adjacency matrix of the induced subgraph where the vertices of X are arranged in a fixed order given by $\sigma \in Sym(X)$. We compare adjacency matrices bitwise line by line. For example, in Figure 1 we have $adj\left(G[X_r], \left(\begin{smallmatrix} u & v & w \\ u & v & w \end{smallmatrix}\right)\right) = \left(\begin{smallmatrix} 0&0&1 \\ 0&0&0 \\ 1&0&0 \end{smallmatrix}\right)$ and $adj\left(G[X_r], \left(\begin{smallmatrix} u & v & w \\ u & w & v \end{smallmatrix}\right)\right) = \left(\begin{smallmatrix} 0&1&0 \\ 1&0&0 \\ 0&0&0 \end{smallmatrix}\right)$

Child bags and their order. In the following two definitions we bring together the logarithmic depth bound of Lemma 2 and what we need for canonization. First, we define an order on split components that is just partially canonical, but which is the basis for our canonization algorithm. Second, we define what we call *valid child bags*, we use them to construct approximate tree decompositions. Note, although we consider graphs of treewidth at most k, we get approximate tree decompositions that allow bags of size at most $4k + 3$.

Definition 1. *Let G be a graph of treewidth at most k and X a bag of size $\leq 4k+3$. Let $G_1, \ldots, G_m$ be its split components. Let $\sigma \in Sym(X)$ be an ordering on the vertices in X. Let $S_1, \ldots, S_l \subseteq X$ be a complete list of minimal separating sets of $G_1, \ldots, G_m$ in X. We define an* order *on $G_1, \ldots, G_m$ with respect to σ using two criteria lexicographically:*

1. *primarily according to σ, it induces a lexicographical order on $S_1, \ldots, S_l$, and*
2. *among them which are equal, we reorder them according to the sizes of their associated split components.*

Note, l is a constant, because $l \leq m$ and in an approximate tree decomposition $l \leq |\mathcal{P}(X)|$ (i.e. $l \leq (4k+3)! \leq 2^{(4k+3)\log(4k+3)}$), that is the number of all possible subsets of X. The primary order of split components is deduced from σ. We give an example: let $\sigma = \left(\begin{smallmatrix} 1&2&3&4 \\ 2&3&1&4 \end{smallmatrix}\right)$ and $S_1 = \{1, 2, 4\}$ and $S_2 = \{3, 4\}$. When sorted according to σ (i.e. $2 < 3 < 1 < 4$), then we have $(2, 1, 4) < (3, 4)$

lexicographically and therefore S_1 comes before S_2. The second criterion brings together split components of equal size, this is useful for canonization and also for the complexity analysis (also see [22]). There are split components which are equal according to Definition 1. To get a default order, they can be sorted according to the label of the minimal vertex in the split components except X. To rearrange them is part of the canonization algorithm.

Definition 2. *Consider G, X, $G_1, \ldots, G_m$ being sorted and σ as in Definition 1. Let $1 \leq p \leq q \leq m$, we consider now $G_p, \ldots, G_q$. Let $S_1, \ldots, S_{l'}$ be all the minimal separating sets of $G_p, \ldots, G_q$. We define three types of* valid child bags of X with respect to split components $G_p, \ldots, G_q$ *as follows:*

(a) Take minimal separating sets of bags: If $l' > 1$ and S_j is the minimal separating set for G_p then the set S_j is a valid child bag.
(b) Take a subset of X and add vertices from G_i: If $l' = 1$ then for $V_i \subseteq X \cup V(G_i)$ with $i \in \{p, \ldots, q\}$ with $|X \cup V_i| \leq 4k+3$ and $\emptyset \subset (X \cap V_i) \subset X$, then the set V_i is a valid child bag.
(c) The set X is a valid child bag.

3.2 Minimal Description for Graphs of Bounded Treewidth

We define a minimal description for graphs. This is based on approximate tree decompositions which are balanced and depth bounded. The minimal description is computed with respect to some parameters.

By Lemma 2 we know that there exist tree decompositions of logarithmic depth, we show that we can guarantee this depth restriction, we use a so called *depth parameter*. We consider perfect trees only.

There is a task where split components are partitioned into classes, e.g. if they have the same size. By Definition 1 split components are ordered and hence, split components in a class can be addressed by an *interval*, e.g. $[p, q]$ for $G_p, \ldots, G_q$. Another parameter is a permutation σ, it describes a unique order of the vertices in the current root bag X.

We consider first some cases concerning the depth parameter. That is, when the total depth is exceeded and we return *no-canon*, or when a minimal description is already computed in the previous step and we just take this.
In the good case the minimal description is a word $C_0C'_0C_1C_2 \in \{0,1,2\}^*$.

Here, C_0 and C'_0 contain information of the root, C_1 and C_2 contain the minimal description of the children. To specify which of two minimal descriptions is the smaller one, we define an order $\prec$ on them.

Definition 3. *We define an* order $\prec$ *on minimal descriptions $C = C_0C'_0C_1C_2$ and $D = D_0D'_0D_1D_2$. We define $C \prec D$ if*

- *$C_0 < D_0$ where we compare adjacency matrices line by line and bit by bit, or*
- *$C_0 = D_0$ but $C'_0 < D'_0$ where we compare not vertex labels according to σ but their positions in the parent of X (example: for $C'_0 = (b, c, d, e)$ and the ordered vertices of* $\mathrm{Parent}(X) = (a, b, c, d)$ *we have $C'_0 = (2, 3, 4)$), or*

– *$C_0 = D_0$ and $C'_0 = D'_0$ but $C_1 \prec D_1$, recursively, or*
– *$C_0 = D_0$ and $C'_0 = D'_0$ and $C_1 = D_1$ but $C_2 \prec D_2$, recursively.*

The construction. Let G be a graph of size n, treewidth at most k, a root bag X of size $\leq 4k+3$, a permutation $\sigma \in Sym(X)$, an interval $[p,q]$ (with $1 \leq p \leq q \leq m$ and $G_1, \ldots, G_m$ the split components in $G \setminus X$ arranged according to Definition 1) and depth parameter $d \in \mathbb{Z}$. A *minimal description* $C(G, X, \sigma, (p,q), d)$ is *no-canon* or a word in $\{0,1,2\}^*$ defined as follows.

If $d < 0$ then $C(G, X, \sigma, (p,q), d) =$ *no-canon*, this tree decomposition exceeds the maximum depth. We call a minimal description *good* if it is different to *no-canon.*

If $d > 0$ and $C(G, X, \sigma, (p,q), d-1) \neq$ *no-canon* then we just copy it from the previous level, that is $C(G, X, \sigma, (p,q), d) = C(G, X, \sigma, (p,q), d-1)$.

If $d = 0$ or $d > 0$ and $C(G, X, \sigma, (p,q), d-1) =$ *no-canon* then $C(G, X, \sigma, (p,q), d) = C_0 C'_0 C_1 C_2$ or *no-canon* defined as follows:

(1) *C_0 encodes the adjacency matrix of the root bag X.*
$C_0 = adj(G[X], \sigma)\{2\}^{(4k+3)^2 - |X|^2}$, the adjacency matrix of the root, filled up with symbol 2, an extra symbol to get the total length $(4k+3)^2$.

(2) *C'_0 encodes the vertices of X ordered by σ.* This part is important for canonization, when bringing all vertices in a unique order.
$C'_0 = v_1 \ldots v_i \{2\}^{\lceil \log(n) \rceil \cdot ((4k+3)-i)}$ with $i = |X|$. The vertices are ordered according to σ, (i.e. $\sigma(v_j) = j$ for all $v_j \in X$). We assume, that the description of every vertex has a fixed length of exactly $\lceil \log n \rceil$ bits.

(3) *Definition of C_1 and C_2 in further subcases.* We only consider the split components with their index inside the interval $[p,q]$, these are $(q-p+1)$ many. Let $S_1, \ldots, S_l \subseteq X$ be all the minimal separating sets (in lexicographical increasing order according to σ) for split components $G_p, \ldots, G_q$ which are ordered according to Definition 1. We partition the split components corresponding to the members in S_1: let $\Theta_1, \ldots, \Theta_t$ be the classes where the corresponding split components have equal size. These *size classes* are arranged in increasing order of the sizes of the corresponding split components. To define C_1 and C_2, we consider the following cases.

(i) $p < q$ **and** $l > 1$**.** That is, we have many such minimal split components, we separate them from the others and recursively canonize them in C_1 and the others in C_2.
Let $G_p, \ldots, G_{q_1}$ be the set of split components separated by S_1. Let ψ be obtained from σ after removing the vertices from $X \setminus S_1$, we define

$$C_1 = C(G, X, \sigma, (p, q_1), d-1) = C(G', S_1, \psi, (1, q_1 - p + 1), d-1)$$

where $G' = G[V(G_p) \cup \cdots \cup V(G_{q_1})]$. Note, since the split components are ordered according to Definition 1, we just take $[p, q_1]$ of $[p,q]$. If $l > 2$ then $C_2 = C(G, X, \sigma, (q_1+1, q), d-1)$.
If $l = 2$ then we define ψ' accordingly as ψ before with respect to S_2,

$$C_2 = C(G, X, \sigma, (q_1+1, q), d-1) = C(G', S_2, \psi', (1, q - q_1), d-1)$$

where $G' = G[V(G_{q_1+1}) \cup \cdots \cup V(G_q)]$.

(ii) $p < q$ **and** $l = 1$ **and** $t > 1$**.** That is, we have a single minimum separating set S_1 but many size classes. We try to find how to partition the size classes which result in the smallest minimal description, i.e. such that we have two sets $\Theta_1, \ldots, \Theta_i$ and $\Theta_{i+1}, \ldots, \Theta_t$.
For each $i \in \{1, \ldots, t-1\}$ let $G_p, \ldots, G_{q_i}$ be the set of split components of size classes $\Theta_1, \ldots, \Theta_i$. We define $C_{1,i} = C(G, X, \sigma, (p, q_i), d-1)$ and $C_{2,i} = C(G, X, \sigma, (q_i + 1, q), d-1)$.
Let (C_{min}, C'_{min}) be the lexicographically smallest pair according to $\prec$ in the set $\{(C_{1,i}, C_{2,i}), (C_{2,i}, C_{1,i}) \mid 1 \leq i \leq t-1\}$. We define $C_1 = C_{min}$ and $C_2 = C'_{min}$.

(iii) $p < q$ **and** $l = 1$ **and** $t = 1$**.** That is, we have many children from one separating set and one size class. We canonize all the children $G_p, \ldots, G_q$ individually and sort their minimal descriptions in ascending order according to $\prec$. Then, we do the same rearrangements with the split components $G_p, \ldots, G_q$. In C_1, we canonize the first half of them and in C_2 the second half. Note, the sorting of $q - p + 1$ children is expensive, hence we reduce the depth parameter logarithmically in the length of this interval.
Let $a = \lceil \log_c(q - p + 1) \rceil$ where $c = 3/2$. For each $i \in \{p, \ldots, q\}$ we define $C'_i = C(G, X, \sigma, (i, i), d - a)$. If one of the $C'_i =$ *no-canon* then $C_1 =$ *no-canon*. Rearrange $C'_p, \ldots, C'_q$ in lexicographical increasing order according to $\prec$. Rearrange $G_p, \ldots, G_q$ according to the new order of $C'_p, \ldots, C'_q$. Let $i = \lceil (q-p)/2 \rceil$. We define $C_1 = C(G, X, \sigma, (p, p+i), d-1)$ and $C_2 = C(G, X, \sigma, (p + i + 1, q), d - 1)$.

(iv) $p = q$**.** That is, we consider one single child. We consider all valid child bags of type (b) in Definition 2. The one which gives the smallest minimal description is selected for C_1, whereas C_2 is filled up with default symbols.
For each set of vertices $V_i \subseteq X \cup G_p$ that is a *valid child bag* as in Definition 2 with $V_i \cap G_p \neq \emptyset$, and each permutation $\psi_{i,j} \in Sym(V_i)$ which is obtained from σ when removing the vertices in $X \setminus V_i$ and adding to the right the vertices in $V_i \setminus X$ in an arbitrary order, we define $C_{1,(i,j)} = C(G_p \setminus (X \setminus V_i), V_i, \psi_{i,j}, (1, m'), d - 1)$
and $(i, j) \in I$, where m' is the number of split components in $G_p \setminus (X \cup V_i)$, i.e. the subgraph $G_p \setminus (X \setminus V_i)$ when removing V_i. We define C_1 to be the minimum of $\bigcup_{(i,j) \in I} C_{1,(i,j)}$. $C_2 = \{2\}^{f(d-1)}$ is a *default description* for a complete binary subtree. We define $f : d \mapsto (|C_0| + |C'_0|) \cdot (2^d - 1)$, i.e. $f(d-1) = \left[(4k+3)^2 + \lceil \log n \rceil \cdot (4k+3)\right] \cdot (2^{(d-1)} - 1)$.

(v) $m = 0$**.** That is, X is not a separating set in G, i.e. a leaf node in a tree decomposition. If no valid child bag exists, then we define $C_1 = C_2 = \{2\}^{f(d-1)}$ with f as in case (iv). That is, both have a default description. Then, a good minimal description is returned.

In general, there is one exception, namely if one of C_1 or C_2 returns *no-canon*, then the minimal description is *no-canon*.
If $C_2 \prec C_1$, then swap them.
If C_1 or C_2 is *no-canon* then $C(G, X, \sigma, (p, q), d) =$ *no-canon*.

The depth of the tree decomposition. We mention some points that have an influence on the depth of the tree decomposition. Note, the construction for $m > 2$ uses some ideas from the proof of a Theorem in a full version of [15], where a binary tree is computed by introducing *white nodes*. The main difference is here, that the whole tree is not given explicitly. Our construction prefers a balanced and binary tree-structure. The depth analysis is inspired from the following more restricted version of Theorem 3.14 in [16].

Lemma 3. (c.f. [16]) *For every tree T with n vertices and height h, there is a binary tree T' of height at most $O(h + \log n)$ such that T can be embedded in T'.*

By Lemma 2 the depth of a tree decomposition that has width $4k + 3$ is at most $c \log_2 n$ for a constant c that depends on k only. Hence, when starting with $d = c' \log_2 n$ (for a constant c') we get a minimal description for $C(G, X, \sigma, (1, m), d)$.

We observe, that our definition guarantees the $O(\log n)$ depth bound. After $O(1)$ steps, the size of the split components is divided at least by 2. In Case (i) we partition the subtrees that belong to the smallest minimal separating set S_1 of X. We can do this, since the number of separating sets is a constant. In Case (ii) we run through all possibilities to split the size classes $\Theta_1, \dots, \Theta_t$ into two sets $\Theta_1, \dots, \Theta_i$ and $\Theta_{i+1}, \dots, \Theta_t$. For example, in two steps we can isolate a size class Θ_i where all the subtrees have together more than half the total size of the subgraph rooted at X: first, split off those to the left and second, those to the right of Θ_i.

In Case (iii) we sort the split components and partition them such that in C_1 at most one more split component is canonized than in C_2. This can only happen when there are more than two split components considered currently. Therefore the size of C_1 is at most 2/3 the size of $C_0 C_0' C_1 C_2$ and this is the reason why $c = 3/2$ in $a = \lceil \log_c q - p + 1 \rceil$. Later in the complexity analysis part (see Lemma 7) we will show that with an inductive argument, split components of size n/i can be canonized by a sub-circuit of depth $c \log(n/i) = c \log n - c \log i$, with c a constant. Hence, to encounter case (iii) recursively is no problem.

In Case (iv) we have a single split component. Hence, we get the following.

Lemma 4. *There is a constant c which depends on k only such that for all graphs G of treewidth at most k, there is a root bag X with m split components in $G \setminus X$, permutation $\sigma \in Sym(X)$ and depth parameter $d = c \log_2 n$, such that the minimal description $C(G, X, \sigma, (1, m), d)$ is* good.

The next is to show that the minimal description is unique up to isomorphism.

Lemma 5. *For a graph G, a constant c, and two bags X, X' with permutations $\sigma \in Sym(X)$ and $\sigma' \in Sym(X')$, with m split components in $G \setminus X$ (and $G \setminus X'$), and depth parameters $d = d' = c \log n$ (for a constant c) it holds that $C(G, X, \sigma, (1, m), d) = C(G, X', \sigma', (1, m), d')$ if and only if there is an automorphism ϕ of G which maps X onto X' via $\sigma(\sigma')^{-1}$.*

A proof can be found in a full version. The main arguments are: an automorphism preserves the edge relation, and hence a tree decomposition. In the construction, we run through all possibilities to define valid child bags. We take a bag as child iff it gives the smallest minimal description.

To obtain a canonical invariant for graphs of treewidth at most k, run through all sets of at most $4k + 3$ vertices as initial root bag X and all permutations $\sigma \in Sym(X)$. According to Lemma 4 there exist good minimal descriptions. We ignore the ones with *no-canon* and select the smallest of all these good minimal descriptions. Thereby, recursively in the parts C'_0, we relabel the vertices according to their first occurrence in the good minimal description.

Theorem 2. *The smallest minimal description of all bags X and permutations σ is a canonical invariant for graphs of treewidth at most k.*

3.3 Complexity Analysis

We prove now that graphs of bounded treewidth can be canonized in AC^1. We construct a circuit which consists of preprocessing steps and a main part, where in $O(\log n)$ levels a minimal description of the input graph is computed.

Valid child bags. We consider Definition 2. We show, that valid child bags can be computed in logspace. For an AC^1-circuit, in a preprocessing step we compute in parallel for each possible bag which are its split components and its valid child bags. A proof is given in a full version.

Lemma 6. *On input of a graph G a bag X with a child bag Y, and an interval $[p, q]$, there is a logspace-computable function that computes whether Y is a valid child bag of X.*

Computing the minimum and sorting. We discuss how to compute the minimal description by an AC^1-circuit. By the recursive construction of minimal descriptions in Section 3.2, the depth of the circuit corresponds to the depth of the tree decompositions. The next lemma is essential in the proof of Theorem 1.

Lemma 7. *Let G be a graph of treewidth at most k, X a root bag, $\sigma \in Sym(X)$ a permutation, $[p, q]$ an interval, d a depth parameter, and suppose we have given minimal descriptions for all valid child bags and all split components $G_p, \ldots, G_q$ for all depth parameters $\leq d - 1$.*

(a) For Case (iii) in the minimal description on Page 217 there is a (depth i)-bounded AC^1-computable function that computes the smallest minimal descriptions with depth parameter $d - i$ for each of the i equal sized split components, and arranges them in lexicographical increasing order.
(b) For all the other cases, there is an AC^0-computable function that computes the smallest minimal description with depth parameter $d - 1$ for each split component.

On input of the minimal descriptions from (a) and (b), there is an AC^0-computable function that computes the minimal description $C(G, X, \sigma, (p, q), d)$.

In the proof, the main tasks are of the following simpler form. Finding the minimum out of N strings of at most N symbols each, can be done in AC^0. Sorting i strings of at most N symbols each can be done by a depth-$O(\log i)$ bounded AC^1 circuit. Finally, we show that the length of a minimal description C depends on the input length n, treewidth k and depth parameter d, i.e.

$$|C| = (|C_0| + |C_0'|) \cdot (2^d - 1) = \left[(4k+3)^2 + (4k+3) \cdot \lceil \log n \rceil\right] \cdot (2^d - 1).$$

For Case (iv) and Case (v), we compute f in a preprocessing step.

We summarize, to get an AC^1-circuit that computes $C(G, X, \sigma, (1, m), d)$, we have preprocessing steps to ensure that the input graph G has treewidth at most k (by Lemma 1), and circuits which compute in parallel for pairs of bags X, Y whether Y is a valid child of X. We have $O(\log n)$ levels of small circuits where minimal descriptions for subtrees are computed, selected or sorted. The total size is polynomial and the total depth of the circuit is $O(\log n)$.

Theorem 3. *There is a constant c and an AC^1-computable function that on input of graph G of treewidth at most k, root bag X of size $\leq 4k+3$, permutation $\sigma \in Sym(X)$, m split components in $G \setminus X$ and depth parameter $d = c \log n$ computes a good minimal description $C(G, X, \sigma, (1, m), d)$ if one exists.*

3.4 The Canonization

The minimal description depends on some parameters: a bag X, a permutation $\sigma \in Sym(X)$ and depth parameter d. There are at most n^{4k+3} many bags and $(4k+3)!$ permutations for X. According to Lemma 4 we can fix $d = c \log_2 n$ (for a constant c) and still get a *good* minimal description. Hence, we set up $n^{4k+3} \cdot (4k+3)!$ many circuits in parallel and compute all possibilities of minimal descriptions. We select in AC^0 the smallest of all these minimal descriptions.

Theorem 4. *There is an AC^1-computable function, that computes a canonical invariant for graphs of treewidth at most k.*

Compute the canonical labeling. To obtain a *canonizing function* the algorithm is doing some extra work in parallel. While computing the minimal description we bring the vertices in a unique order. For this, in a minimal description $C = C_0 C_0' C_1 C_2$ the part C_0' plays a central role.

The minimal description gives an order to the bags. We list now the vertices of the bags in a fixed order and the inserted vertices of each bag with their original vertex names.

To define the canonizing function, we use a simple logspace-computable procedure which can be found e.g. in [12]. The order of the occurrences of all vertices defines a fixed order. After renaming the vertices according to this order we arrange the edges in lexicographical increasing order. Note, the renaming of the vertices is an automorphism from G onto its canon, i.e. this is a *canonical labeling*. Hence, the canonization of graphs of treewidth at most k is in AC^1. This completes the proof of Theorem 1.

Conclusion. We improve the upper bound of the canonization problem for bounded treewidth graphs very close to the LogCFL upper bound of isomorphism testing [11]. However, it is not clear how to improve the new AC^1 upper bound with known standard techniques for canonization. In [15] interesting concepts for bounded treewidth graphs are introduced, they state the question whether these can be used for isomorphism testing or canonization.

Acknowledgment. We thank Jacobo Torán and anonymous referees for comments and helpful discussions.

References

1. Arnborg, S., Corneil, D., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM Journal on Algebraic and Discrete Methods 8(2), 277–284 (1987)
2. Arvind, V., Das, B., Köbler, J.: A logspace algorithm for partial 2-tree canonization. In: Hirsch, E.A., Razborov, A.A., Semenov, A., Slissenko, A. (eds.) Computer Science – Theory and Applications. LNCS, vol. 5010, pp. 40–51. Springer, Heidelberg (2008)
3. Babai, L., Luks, E.M.: Canonical labeling of graphs. In: 15th Annual ACM Symposium on Theory of Computing (STOC), pp. 171–183 (1983)
4. Bodlaender, H.L.: NC-algorithms for graphs with small treewidth. In: Proceedings of the 14th International Workshop on Graph-Theoretic Concepts in Computer Science (WG), pp. 1–10 (1989)
5. Bodlaender, H.L.: Polynomial algorithms for graph isomorphism and chromatic index on partial k-trees. Journal of Algorithms 11, 631–644 (1990)
6. Bodlaender, H.L.: A linear-time algorithm for finding tree-decompositions of small treewidth. SIAM Journal on Computing 25(6), 1305–1317 (1996)
7. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theoretical Computer Science 209, 1–45 (1998)
8. Bodlaender, H.L., Hagerup, T.: Parallel algorithms with optimal speedup for bounded treewidth. SIAM Journal on Computing 27(6), 1725–1746 (1998)
9. Bodlaender, H.L., Koster, A.M.: Combinatorial optimization on graphs of bounded treewidth. The Computer Journal 51(3), 255–269 (2008)
10. Chandrasekharan, N., Hedetniemi, S.T.: Fast parallel algorithms for tree decomposition and parsing partial k-trees. In: proceedings of the 26th Annual Allerton Conference on Communication, Control, and Computing, pp. 283–292 (1988)
11. Das, B., Torán, J., Wagner, F.: Restricted space algorithms for isomorphism on bounded treewidth graphs. In: Proceedings of the 27th International Symposium on Theoretical Aspects of Computer Science, pp. 227–238 (2010)
12. Datta, S., Limaye, N., Nimbhorkar, P.: 3-connected planar graph isomorphism is in log-space. In: Proceedings of the 28th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp. 153–162 (2008)
13. Datta, S., Limaye, N., Nimbhorkar, P., Thierauf, T., Wagner, F.: Planar graph isomorphism is in log-space. In: Annual IEEE Conference on Computational Complexity (CCC), pp. 203–214 (2009)
14. Datta, S., Nimbhorkar, P., Thierauf, T., Wagner, F.: Isomorphism for $K_{3,3}$-free and K_5-free graphs is in log-space. In: Proceedings of the 29th Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), pp. 145–156 (2009)

15. Elberfeld, M., Jakoby, A., Tantau, T.: Logspace versions of the theorems of Bodlaender and Courcelle. In: Proceedings of the 51st Annual Symposium on Foundations of Computer Science, FOCS (2010)
16. Elberfeld, M., Jakoby, A., Tantau, T.: Logspace versions of the theorems of Bodlaender and Courcelle. Technical Report TR10-062, Electronic Colloquium on Computational Complexity, ECCC (2010)
17. Grohe, M., Verbitsky, O.: Testing graph isomorphism in parallel by playing a game. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 3–14. Springer, Heidelberg (2006)
18. Köbler, J.: On graph isomorphism for restricted graph classes. In: Beckmann, A., Berger, U., Löwe, B., Tucker, J.V. (eds.) CiE 2006. LNCS, vol. 3988, pp. 241–256. Springer, Heidelberg (2006)
19. Köbler, J., Kuhnert, S.: The isomorphism problem for k-trees is complete for logspace. In: Královič, R., Niwiński, D. (eds.) MFCS 2009. LNCS, vol. 5734, pp. 537–548. Springer, Heidelberg (2009)
20. Köbler, J., Verbitsky, O.: From invariants to canonization in parallel. In: Hirsch, E.A., Razborov, A.A., Semenov, A., Slissenko, A. (eds.) Computer Science – Theory and Applications. LNCS, vol. 5010, pp. 216–227. Springer, Heidelberg (2008)
21. Lagergren, J.: Efficient parallel algorithms for tree-decomposition and related problems. In: In proceedings of the 31st Annual Symposium on Foundations of Computer Science (FOCS), pp. 173–182 (1990)
22. Lindell, S.: A logspace algorithm for tree canonization (extended abstract). In: Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC), pp. 400–404. ACM, New York (1992)
23. Luks, E.M.: Permutation groups and polynomial-time computation. DIMACS series in Discrete Mathematics and Theoretical Computer Science 11, 139–175 (1993)
24. Reed, B.A.: Finding approximate separators and computing tree width quickly. In: Proceedings of the 24th Annual ACM Symposium on Theory of Computing (STOC), pp. 221–228 (1992)
25. Robertson, Seymour: Graph minors. II. algorithmic aspects of tree-width. Journal of Algorithms (ALGORITHMS) 7(3), 309–322 (1986)
26. Wanke, E.: Bounded tree-width and LOGCFL. Journal of Algorithms 16 (1994)

Snakes and Cellular Automata: Reductions and Inseparability Results*

Jarkko Kari

Department of Mathematics, University of Turku, FI-20014 Turku, Finland
jkari@utu.fi

Abstract. A careful analysis of an old undecidability proof reveals that periodicity and non-surjectivity of two-dimensional cellular automata are recursively inseparable properties. Analogously, Wang tile sets that admit tilings of arbitrarily long loops (and hence also infinite snakes) are recursively inseparable from the tile sets that admit no loops and no infinite snakes. The latter inseparability result actually implies the first one in a trivial way.

1 Introduction

Tilings and two-dimensional cellular automata are closely related concepts. Tilings of $\mathbb{Z}^2$ are static objects where a local, position invariant matching relation specifies which patterns of symbols are allowed. These are also known as two-dimensional subshifts of finite type. Cellular automata, in contrast, are dynamic counterparts determined by a local, position invariant update rule of the symbols. Cellular automata are continuous in the standard product topology, and hence are endomorphisms of the full (two-dimensional) shift.

The tiling problem is the decision problem that asks if given tiles admit at least one valid tiling of the plane. The problem was proved undecidable by R. Berger in 1966 [4]. The tiling problem turns out to be very useful in establishing undecidability results concerning cellular automata. In [8,9] the tiling problem was reduced into the decision problem that asks whether a given two-dimensional cellular automaton is reversible, that is, admits an inverse rule that retraces the evolution back in time. The reduction is based on a complex set of tiles with arrows on them that satisfies a plane-filling property: If one moves from tile-to-tile, following the arrows, and if one never sees a tiling error, then the trajectory is forced to be plane-filling in the sense that arbitrarily large squares are fully covered by it.

Using a similar construction, it was also shown in [9] that it is undecidable whether a given two-dimensional cellular automaton is surjective, that is, has no Garden-of-Eden configurations. These are configurations without a pre-image. The surjectivity problem was later shown by B. Durand to have a simpler undecidability proof, where the complex tiles of [9] were replaced by much simpler

* Research supported by the Academy of Finland Grant 131558.

ones [5]. One observation that we want to point out in the present talk is that – even though it has been superseded by a better one – the original undecidability proof has the advantage that the undecidability proofs for surjectivity and reversibility can be combined into a single reduction that shows the following inseparability result.

Theorem 1. *The classes of*

(a) non-surjective,
(b) periodic,

two-dimensional cellular automata are recursively inseparable.

The theorem states that any decidable class of cellular automata that contains all periodic cellular automata must also contain some non-surjective ones. As a direct corollary to we see that properties such as injectivity and surjectivity on periodic configurations, openness and surjectivity on q-finite configurations are undecidable among two-dimensional cellular automata, since all these properties separate reversible from non-surjective automata.

The plane-filling tiles of [8,9] have found new applications in algorithmic questions that arise in the context of the tiling model of self-assembly. In this model, Wang tiles – unit square tiles with colored edges – stick to each other to form patches [1,16]. A new tile may stick to a patch if the colors on the adjacent edges match, and if the total strength of the bonds – given by a numerical value associated to each color – exceeds some threshold value. Algorithmic questions concerning the model arise. For example, the termination problem asks to determine, for a given a set of tiles, whether the assembly process necessarily reaches a terminal patch in which no new tiles can stick, or is unbounded growth possible. It turns out that the termination problem is undecidable [2,3], even under the following two restricted types of non-numeric assembly rules: Under the weak sticking, a new tile may attach to a patch whenever one of the common edges has the same color. Under the strong sticking, a tile may attach to a patch only if all common edges between the tile and the patch have matching colors. The undecidability proofs in [2,3] are reductions from the tiling problem, using the plane filling tiles in a similar fashion as in [8,9].

It is easy to see that the termination problem is, in fact, equivalent to the infinite snake tiling problem. This question asks whether it is possible to form from the given Wang tiles an infinite, non-self-intersecting sequence on the plane, where consecutive tiles are placed in adjacent positions. In a strong snake (termed a *zipper* in [2,3]) any two neighboring tiles of the snake are required to have a common color on their shared edge, while in a weak snake (called a *ribbon* in [2,3]) only the colors of the consecutive tiles are required to match (and where the snake returns back to touch an earlier tile, matching is not required). The decidability status of the (weak) infinite snake tiling problem was actually proposed independently as an open problem already in [6].

One can also ask a similar question about the possibility to form a loop on the plane using the given tiles. Again, the question can be formulated under the strong and under the weak matching rules, depending on whether colors are required to match also between neighbors that are not consecutive in the loop. The weak and the strong loop tiling problems were proved undecidable in [10] using a construction similar to [5]. One can prove the loop formation undecidable also using the more complex plane-filling tiles from [9]. Again, this approach has the advantage that the infinite snake tiling problem and the loop tiling problem can be treated in the same reduction, so that – in an exact analogy to the reversibility and surjectivity problems of two-dimensional cellular automata – we obtain the following inseparability result.

Theorem 2. *The classes of Wang tile sets that*

(a) admit tilings of arbitrarily long loops (and hence also infinite snakes),
(b) do not admit any tilings of loops or infinite snakes,

are recursively inseparable. This inseparability holds even if in (a) we require strong matching while in (b) the weak matching may be used.

In Section 2 we recall the relevant definitions on cellular automata, tilings, plane-filling paths and snake tiling problems. Section 3 contains proofs of Theorems 1 and 2.

2 Definitions

Let us call the elements of $\mathbb{Z}^2$ *cells.* Assignments $c : \mathbb{Z}^2 \longrightarrow S$ of symbols from a finite set S to cells are called *configurations.* The set of all configurations over S is denoted by $S^{\mathbb{Z}^2}$. A *neighborhood vector* of size m is an m-tuple $N = (\boldsymbol{n}_1, \boldsymbol{n}_2, \ldots, \boldsymbol{n}_m)$ where $\boldsymbol{n}_i \in \mathbb{Z}^2$, $i = 1, 2 \ldots, m$, specify the relative offsets from each cell to its neighbors: The *neighbors* of cell $\boldsymbol{n} \in \mathbb{Z}^2$ are the cells $\boldsymbol{n}+\boldsymbol{n}_1, \boldsymbol{n}+\boldsymbol{n}_2, \ldots, \boldsymbol{n}+\boldsymbol{n}_m$. We assume that $\boldsymbol{n}_i \neq \boldsymbol{n}_j$ for $i \neq j$ so that each neighbor only appears once in the list.

2.1 Tilings

A *tile set* is triplet (T, N, R) where T is finite set whose elements are the *tiles*, N is a neighborhood vector of size m and $R \subseteq T^m$ is a relation that specifies *forbidden patterns.* Tiling is valid in position $\boldsymbol{n} \in \mathbb{Z}^2$ of a configuration $t \in T^{\mathbb{Z}^2}$ if the pattern in position $\boldsymbol{n}$ is not forbidden, that is, if

$$[t(\boldsymbol{n}+\boldsymbol{n}_1), t(\boldsymbol{n}+\boldsymbol{n}_2), \ldots, t(\boldsymbol{n}+\boldsymbol{n}_m)] \notin R.$$

A configuration is a *valid tiling* if it is valid in all positions $\boldsymbol{n} \in \mathbb{Z}^2$. Since N and R are usually clear from the context, we often simply refer to T as the tile set.

Wang tiles are particular kind of tiles where the matching condition is given by coloring of the edges of unit square tiles. Two adjacent tiles *stick* if the colors on the abutting edges are the same. A configuration is a valid tiling if all pairs of adjacent tiles stick. The following two decision problems concerning Wang tiles are used in Section 3 to obtain our inseparability results.

The *domino problem* is the decision problem to determine if a given Wang tile set admits a valid tiling. The domino problem is undecidable [4]. Let $L \subseteq \Sigma^*$ be a recursively enumerable language over alphabet Σ. There is an effective reduction that produces, for any given $w \in \Sigma^*$, a Wang tile set T that admits a valid tiling if and only if $w \notin L$.

Let $b \in T$. A *b-finite configuration* over T is any $t \in T^{\mathbb{Z}^2}$ whose b-support $\{\boldsymbol{n} \in \mathbb{Z}^2 \mid t(\boldsymbol{n}) \neq b\}$ is finite. A *b-uniform* configuration t assigns $t(\boldsymbol{n}) = b$ for all $\boldsymbol{n} \in \mathbb{Z}^2$. A Wang tile b is called *blank* if the b-uniform configuration is a valid tiling. In the *finite tiling problem* one is given a Wang tile set T and a blank tile $b \in T$. The question is to determine whether there exists a valid b-finite tiling that is not b-uniform. This problem is easily seen to be undecidable [9] using the techniques developed by H.Wang [15]. If $K \subseteq \Sigma^*$ is recursively enumerable then there is an effective reduction that produces, for any given $w \in \Sigma^*$, a Wang tile set T and $b \in T$ such that a non-uniform, valid b-finite tiling exists if and only if $w \in K$.

2.2 Cellular Automata

A *cellular automaton (CA)* is a triplet (S, N, f) where S is a finite *state set*, N is a neighborhood vector of size m, and $f : S^m \longrightarrow S$ is an *update rule* that determines the new state of each cell based on the pattern in its neighborhood. The CA defines a transformation $F : S^{\mathbb{Z}^2} \longrightarrow S^{\mathbb{Z}^2}$ of the configuration space where $F(c) = e$ when

$$\forall \boldsymbol{n} \in \mathbb{Z}^2 : \qquad e(\boldsymbol{n}) = f[c(\boldsymbol{n}+\boldsymbol{n}_1), c(\boldsymbol{n}+\boldsymbol{n}_2), \ldots, c(\boldsymbol{n}+\boldsymbol{n}_m)].$$

Transformation F is called a *CA function.* It is well known that CA functions are precisely those transformations of the configuration space that are translation invariant and continuous in the product topology [7].

If F is a bijection then the inverse F^{-1} is always a CA function [7]. In this case the CA is called *reversible.* A CA is called *injective* (*surjective*) if F is one-to-one (onto, respectively). Let us call a pair $c, e \in S^{\mathbb{Z}^d}$ of configurations *asymptotic* if the set $\{\boldsymbol{n} \in \mathbb{Z}^2 \mid c(\boldsymbol{n}) \neq e(\boldsymbol{n})\}$ of cells where they differ is finite. Function F is *pre-injective* if for all asymptotic $c, e \in S^{\mathbb{Z}^2}$ such that $c \neq e$, we have $F(c) \neq F(e)$. The important *Garden-of-Eden*-theorem states that a CA function is surjective if and only if it is pre-injective [13,14]. In particular, this implies that injective CA are automatically also surjective, and hence reversible. See [11] for more details on the basic concepts and results on cellular automata.

2.3 Directed Tiles and Paths

Let us associate to each tile a pointer to a neighbor. In the following these *follower vectors* take values from the set $\{(\pm 1, 0), (0, \pm 1)\}$, that is, the vector points to one of the four adjacent neighbors of a cell. We obtain a *directed* tile set (T, N, R, d) where (T, N, R) is a tile set and $d : T \longrightarrow \{(\pm 1, 0), (0, \pm 1)\}$ gives the follower vector of each tile.

Let $t \in T^{\mathbb{Z}^2}$ be a configuration of directed tiles. An infinite sequence $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots$ of cells $\boldsymbol{p}_i \in \mathbb{Z}^2$ is a *directed path* (or simply a *path*) on t if $\boldsymbol{p}_{i+1} = \boldsymbol{p}_i + d(t(\boldsymbol{p}_i))$ for all $i \in \mathbb{N}$. Positions on the path hence form a succession where the offset to the next position is always given by the follower vector of the tile in the previous position. For any $n \in \mathbb{N}_0$ and $(x, y) \in \mathbb{Z}^2$, let us denote by

$$S^n_{x,y} = \{x - n, \ldots, x + n\} \times \{y - n, \ldots, y + n\}$$

the square of size $(2n + 1) \times (2n + 1)$ centered at position (x, y). We say that directed path $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots$ is *plane-filling* if it covers arbitrarily large squares, that is if, for all $n \in \mathbb{N}$, there exists $(x, y) \in \mathbb{Z}^2$ such that $S^n_{x,y} \subseteq \{\boldsymbol{p}_i \mid i \in \mathbb{N}\}$.

In [8,9] a directed tile set Snakes is constructed that satisfies the following *plane-filling property*: Consider a directed path $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots$ on an arbitrary configuration t. Then either

(i) for some $i \in \mathbb{N}$ the tiling in t is not valid in position $\boldsymbol{p}_i$, or
(ii) the path is plane-filling.

Moreover, Snakes admits valid tilings. This tile set was used in reducing the domino problem into the problem of determining whether a given two-dimensional cellular automaton is reversible.

In [2,3] the construction is further improved for the purpose of applying it to snake tilings. A set of directed Wang tiles is presented such that any directed path $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots$ on any configuration t satisfies the following: Either

(i) for some $i, j \in \mathbb{N}$ the positions $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ are adjacent but the corresponding Wang tiles $t(\boldsymbol{p}_i)$ and $t(\boldsymbol{p}_j)$ do not stick, or
(ii) the path is plane-filling.

In the improved Wang tile set, any infinite path that follows the directions must be plane-filling if it does not contain a mismatch between two tiles belonging to the path. This requires more than the original construction of Snakes where the plane-filling path was forced to be formed under the stronger assumption that there is no mismatch within the neighborhood of any tile of the path – and this neighborhood may contain positions outside the path. The improvement was established by going into a higher block presentation of the tiles.

In [9] the set Snakes is further modified for the purpose of proving surjectivity of CA undecidable. For this goal, the paths are allowed to form loops that cover squares with recognizable borders and centers. The higher block conversion can be performed also on this set, analogously to [2,3], yielding a set of directed Wang tiles Loops with the following property.

Property 1. The directed tile set Loops is partitioned into three disjoint sets A, B and C. Let $t \in T^{\mathbb{Z}^2}$ and let $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots$ be a directed path on t. Then one of the following three cases holds:

(i) For some $i, j \in \mathbb{N}$ the positions $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ are adjacent but the corresponding Wang tiles $t(\boldsymbol{p}_i)$ and $t(\boldsymbol{p}_j)$ do not stick, or
(ii) the path is plane-filling, or
(iii) the path is a loop that covers some $(2n+1) \times (2n+1)$ -square with A-tiles on the boundary, a single B-tile at the center, and C-tiles elsewhere:

$$\exists k \geq 1, \forall i \in \mathbb{N} : \quad \boldsymbol{p}_{i+k} = \boldsymbol{p}_i,$$
$$\exists n \in \mathbb{N}, x, y \in \mathbb{Z} : \; S^n_{x,y} \subseteq \{\boldsymbol{p}_i \mid i \in \mathbb{N}\} \text{ and } \begin{cases} \forall \boldsymbol{p} \in S^n_{x,y} \setminus S^{n-1}_{x,y} : \; t(\boldsymbol{p}) \in A, \\ t(x,y) \in B, \\ \forall \boldsymbol{p} \in S^{n-1}_{x,y} \setminus \{(x,y)\} : \; t(\boldsymbol{p}) \in C. \end{cases}$$

Moreover, there are valid tilings with paths that satisfy (ii), and for every $n_0 \in \mathbb{N}$ there are valid tilings with paths that satisfy (iii) for some $n \geq n_0$.

In Section 3 the tile set Loops is used to reduce the domino problem and the finite tiling problem to prove inseparability results.

2.4 Snake Tiling Problems

Let T be a set of (directed) Wang tiles. In this section we define two variants of infinite (directed) snakes. We call these strong and weak snakes, respectively. We also define two analogous variants of (directed) loops.

Let us first consider the case when the tiles in T are directed. A *strong infinite directed snake* is a directed path $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots$ on some configuration $t \in T^{\mathbb{Z}^2}$ such that

(a) the path does not cross itself: $\boldsymbol{p}_i \neq \boldsymbol{p}_j$ for $i \neq j$, and
(b) There is no tiling error between any adjacent tiles on the path: for any $i, j \in \mathbb{N}$, if the positions $\boldsymbol{p}_i$ and $\boldsymbol{p}_j$ are adjacent then the corresponding Wang tiles $t(\boldsymbol{p}_i)$ and $t(\boldsymbol{p}_j)$ stick.

Term "directed zipper" was used in [2,3] for strong infinite directed snakes. Note that *Property 1* states that all strong infinite directed snakes by Loops are plane-filling.

A *weak infinite directed snake* is defined analogously but condition (b) above is replaced by the weaker requirement that consecutive tiles on the path stick:

(b') for all $i \in \mathbb{N}$, tiles $t(\boldsymbol{p}_i)$ and $t(\boldsymbol{p}_{i+1})$ stick.

These were called "directed ribbons" in [2,3].

The undirected variants are defined by allowing the path to continue from a cell to any of its four adjacent neighbors. An *undirected path* is any sequence $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots$ of positions such that $\boldsymbol{p}_i$ and $\boldsymbol{p}_{i+1}$ are adjacent, for all $i \in \mathbb{N}$. Let T be a set of undirected Wang tiles. *Strong infinite snakes* and *weak infinite snakes* are defined analogously to the directed variants, using undirected paths instead of directed ones.

The (strong or weak) infinite snake tiling problems ask one to determine, for a given (possibly directed) Wang tile set, whether a (strong or weak, respectively) infinite snake exists. That is: is it possible to place tiles on the plane to form an infinite succession of matching tiles. The difference between the strong and weak variants is whether matching is required also between the adjacent tiles that are not consecutive on the snake. All variants were proved undecidable in [2,3].

In [10] questions concerning loop formation were asked. For $k > 2$, a *directed loop* of length k on configuration t is a finite initial segment $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_k$ of a directed path on t in which $\boldsymbol{p}_{k+1} = \boldsymbol{p}_1$ while $\boldsymbol{p}_i \neq \boldsymbol{p}_j$ for $1 \leq i < j \leq k$. An *undirected loop* is defined analogously from an undirected path, when the Wang tiles have no directions. A *strong (directed) loop* is a (directed) loop in which adjacent tiles match in color, while in a *weak (directed) loop* only consecutive tiles of the loop (as well as the last and the first tile) are required to stick. It was shown in [10] that it is undecidable is to determine if a given (directed or undirected) tile set admits strong or weak loops.

By a standard compactness argument, the existence of arbitrarily long (weak or strong) loops implies the existence of (weak or strong, respectively) infinite snakes.

3 Inseparability Results

Let $L, K \subseteq \Sigma^*$ be two disjoint recursively enumerable languages over some alphabet Σ that are *recursively inseparable.* This means that there is no recursive set $D \subseteq \Sigma^*$ such that $L \subseteq D$ and $K \cap D = \emptyset$. Such sets are well known to exist [12].

Because L is recursively enumerable one can, using the reduction method of [4], effectively construct for any given word $w \in \Sigma^*$ a set T_1 of Wang tiles that admits a valid tiling of the plane if and only if $w \notin L$.

Because K is recursively enumerable one can, using the reduction method used in [9], effectively construct for any given word $w \in \Sigma^*$ a set T_2 of Wang tiles and $b \in T_2$ such that there is a b-finite, non-uniform tiling of the plane if and only if $w \in K$.

For any given $w \in \Sigma^*$, one can hence effectively construct the following directed Wang tile set $T_w \subseteq T_1 \times T_2 \times$ Loops, where Loops is the directed Wang tile set that satisfies *Property 1.* The color matching in T_w is checked in all three components of the "sandwich" -tiles. The follower vector of a tile in T_w is given by the follower vector in its Loops -component. We take into T_w all triplets $(s_1, s_2, l) \in T_1 \times T_2 \times$ Loops under the following constraints:

(a) if $l \in A$ then $s_2 = b$, and
(b) if $l \in B$ then $s_2 \neq b$.

Here $A, B \subseteq$ Loops refer to the subsets described in *Property 1.* The constraints guarantee that a correctly tiled square with A-tiles on the boundary and a B-tile in the center is possible only if T_2 admits a b-finite, non-uniform tiling. The construction proves the following claim:

Proposition 1. *The following two classes of directed tile sets T are recursively inseparable:*

(a) Sets T that admit valid tilings with arbitrarily long directed loops.
(b) Sets T that do not admit any strong directed loops or any strong directed infinite snakes.

Proof. We use the sandwich tile set T_w above, constructed effectively for any given $w \in \Sigma^*$.

1) Suppose $w \in L$. Then T_1 does not admit a tiling of the plane, and T_2 does not admit a b-finite, non-uniform tiling of the plane. Consider an arbitrary configuration $t \in T_w^{\mathbb{Z}^2}$ and a directed path $P = \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots$ on t. By *Property 1* of Loops, either

(i) P contains a color mismatch in the Loops component between two adjacent tiles on P, or
(ii) P is plane-filling, or
(iii) P is a loop that covers some square with A-tiles on the boundary and a B-tile at the center.

But in case (ii) there must be a mismatch in the T_1-components of two tiles on the path because otherwise T_1 would admit valid tilings of arbitrarily large squares. And in case (iii) there must be a mismatch in the T_2-components, because otherwise T_2 would admit a tiling of a square with b's on the boundary and some non-b tile in the center. We conclude that every directed path must contain a color mismatch between two of its tiles, and hence T_w belongs to class (b) in the statement of the theorem.

2) Suppose $w \in K$. Then T_1 admits a tiling t_1 of the plane, and T_2 admits a b-finite, non-uniform tiling t_2 of the plane. Consider a configuration t over T_w whose first and second layers consist of these valid tilings t_1 and t_2, respectively. The third layer of t contains a correct tiling by Loops, with a directed loop that covers a $(2n+1) \times (2n+1)$ square $S^n_{x,y}$. By *Property 1*, this square may be chosen for arbitrarily large values of n, and positioned in such a way that $t_2(x, y) \neq b$ and $t_2(\boldsymbol{p}) = b$ for all $\boldsymbol{p} \in S^n_{x,y} \setminus S^{n-1}_{x,y}$. We conclude that there are valid tilings with arbitrarily long correctly tiled loops, and T_w is in class (a).

The two cases above prove the claimed recursive inseparability. Any recursive separation of (a) and (b) would namely provide a recursive separation between languages L and K. □

Proof of Theorem 2. Proposition 1 directly implies Theorem 2 using the motif-construction of [2,3]. In this construction, directed tiles are effectively replaced by a sequence of undirected "minitiles" that only can stick to form contours of large squares with suitable bumps and dents along the boundaries that simulate the colors of the original tiles. For any given directed Wang tile set T, this effective construction provides a set U of undirected Wang tiles such that

$$\begin{array}{ccc} T \text{ admits strong directed loops} & \qquad & T \text{ admits strong directed infinite snakes} \\ \Updownarrow & & \Updownarrow \\ U \text{ admits strong loops} & & U \text{ admits strong infinite snakes} \\ \Updownarrow & & \Updownarrow \\ U \text{ admits weak loops} & & U \text{ admits weak infinite snakes} \end{array}$$

Moreover, if the loops admitted by T are arbitrarily long, so are the loops admitted by U. This reduction clearly takes classes (a) and (b) of Proposition 1 into the classes (a) and (b) of Theorem 2, respectively.

Proof of Theorem 1. This is a direct reduction from Theorem 2: For any undirected Wang tile set T we associate a two-dimensional cellular automaton F with the state set $T \times \{\uparrow, \rightarrow, \downarrow, \leftarrow\} \times \{0,1\}$. The local update rule keeps the T-component and the arrow-component of a state unchanged. The bit-component may be changed. A cell $\boldsymbol{n}$ is active if and only if its T-component sticks to the T-component of its follower cell. The follower cell is simply the adjacent cell in the direction of the arrow at $\boldsymbol{n}$. An inactive cell does not change its state, while an active cells replaces its bit x by $x \oplus y$, where y is the bit of the follower and $\oplus$ denotes the modulo two addition of the bits.

If T is in class (a) of Theorem 2 then the CA F is not surjective: Consider a configuration $t \in T^{\mathbb{Z}^2}$ that contains a valid weak loop. Assign the cells of the loop the arrows that point to next cell of the loop. All cells of the loop are then active. Because F executes the XOR CA along the loop, it is clear that a CA configuration with an odd number of 1's on the loop has no pre-image. The CA is not surjective.

If T is in class (b) of Theorem 2 then the CA F is reversible, and even periodic: Any path in any configuration over T contains a tiling error between some consecutive tiles. By compactness this means that there is a uniform bound n such that for any path $\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots$ on any configuration $t \in T^{\mathbb{Z}^2}$, there is a mismatch between consecutive tiles $t(\boldsymbol{p}_i)$ and $t(\boldsymbol{p}_{i+1})$, for some $i \leq n$. Cell $\boldsymbol{p}_i$ is inactive, and it easily follows that CA F is 2^n-periodic.

4 Conclusions

We have shown the recursive inseparability of two-dimensional cellular automata that are periodic from those that are not surjective. It remains an open challenge to make these classes smaller. In particular, it seems likely that among non-surjective CA there would be interesting proper subclasses that are inseparable from the periodic cellular automata. For example, one wonders whether one can recursively separate the periodic CA from the CA that are ultimately periodic but not periodic.

References

1. Adleman, L.: Towards a mathematical theory of self-assembly. Technical Report 00-722, Department of Computer Science, University of Southern California (2000)
2. Adleman, L., Kari, J., Kari, L., Reishus, D.: On the decidability of self-asssembly of infinite ribbons. In: Proceedings of FOCS 2002, 43rd Annual Symposium on Foundations of Computer Science, pp. 530–537 (2002)
3. Adleman, L., Kari, J., Kari, L., Reishus, D., Sosík, P.: The Undecidability of the Infinite Ribbon Problem: Implications for Computing by Self-Assembly. SIAM Journal on Computing 38(6), 2356–2381 (2009)

4. Berger, R.: The undecidability of the domino problem. Mem. Amer. Math. Soc. 66 (1966)
5. Durand, B.: The surjectivity problem for 2D Cellular Automata. Journal of Computer and System Sciences 49, 149–182 (1994)
6. Etzion-Petruschka, Y., Harel, D., Myers, D.: On the solvability of domino snake problems. Theoretical Computer Science 131, 243–269 (1994)
7. Hedlund, G.: Endomorphisms and automorphisms of shift dynamical systems. Mathematical Systems Theory 3, 320–375 (1969)
8. Kari, J.: Reversibility of 2D cellular automata is undecidable. Physica D 45, 379–385 (1990)
9. Kari, J.: Reversibility and surjectivity problems of cellular automata. Journal of Computer and System Sciences 48, 149–182 (1994)
10. Kari, J.: Infinite Snake Tiling Problems. In: Ito, M., Toyama, M. (eds.) DLT 2002. LNCS, vol. 2450, pp. 67–77. Springer, Heidelberg (2003)
11. Kari, J.: Theory of Cellular Automata: a survey. Theoretical Computer Science 334, 3–33 (2005)
12. Kleene, S.: A symmetric form of Gödel's theorem. Proceedings of the Koninklijke Nederlandse Akademie van Wetenschappen, Series A 53, 800–802 (1950); Also inIndagationes Mathematicae 12, 244–246 (1950)
13. Moore, E.F.: Machine Models of Self-reproduction. In: Proceedings of the Symposium in Applied Mathematics, vol. 14, pp. 17–33 (1962)
14. Myhill, J.: The Converse to Moore's Garden-of-Eden Theorem. Proceedings of the American Mathematical Society 14, 685–686 (1963)
15. Wang, H.: Proving theorems by pattern recognition. II. Bell Systems Technical Journal 40, 1–42 (1961)
16. Winfree, E.: Algorithmic Self-Assembly of DNA. Ph.D. thesis, California Institute of Technology, Pasadena, CA (1998)

Computing the Clique-Width of Large Path Powers in Linear Time via a New Characterisation of Clique-Width

Pinar Heggernes[1], Daniel Meister[2], and Udi Rotics[3]

[1] Department of Informatics, University of Bergen, Norway
pinar.heggernes@ii.uib.no
[2] Theoretical Computer Science, University of Trier, Germany
daniel.meister@uni-trier.de
[3] Netanya Academic College, Netanya, Israel
rotics@netanya.ac.il

Abstract. Clique-width is one of the most important graph parameters, as many NP-hard graph problems are solvable in linear time on graphs of bounded clique-width. Unfortunately, the computation of clique-width is among the hardest problems. In fact, we do not know of any other algorithm than brute force for the exact computation of clique-width on any large graph class of unbounded clique-width. Another difficulty about clique-width is the lack of alternative characterisations of it that might help in coping with its hardness. In this paper, we present two results. The first is a new characterisation of clique-width based on rooted binary trees, completely without the use of labelled graphs. Our second result is the exact computation of the clique-width of large path powers in polynomial time, which has been an open problem for a decade. The presented new characterisation is used to achieve this latter result. With our result, large k-path powers constitute the first non-trivial infinite class of graphs of unbounded clique-width whose clique-width can be computed exactly in polynomial time.

1 Introduction

Clique-width is a graph parameter that has many algorithmic applications [5]. In particular, NP-hard graph problems that are expressible in a certain type of monadic second-order logic admit algorithms with linear running time on graphs whose clique-width is bounded by a constant [6, 22]. Unfortunately, it is NP-hard to compute the clique-width of a given graph [9]. Fellows et al. ask whether the computation of clique-width is fixed-parameter tractable when parametrised by the clique-width of the input graph [9]. This question is still open. Furthermore, we do not know of an algorithm with running time c^n, where c is a constant. Although clique-width has received a lot of attention recently [2, 4, 8, 9, 14, 15, 17–19, 21], positive results known on the computation of clique-width so far are very restricted. Graphs of clique-width at most 3 can be recognised in polynomial time, and their exact clique-width can be computed efficiently [3]. Examples of such graph classes are cographs [7], trees and distance-hereditary graphs [10]. Regarding classes of unbounded clique-width, the class of square grids is the only class for which a polynomial-time clique-width computation algorithm is known [10].

Clique-width is often compared to treewidth, as the same set of problems that are efficiently solvable on graphs of bounded clique-width are also efficiently solvable on graphs of bounded treewidth. However, clique-width is more general than treewidth, as graphs of bounded treewidth have bounded clique-width [4, 7], whereas there are graph classes of bounded clique-width whose treewidth is not bounded (for example complete graphs). For such graph classes, the gap between the two parameters may be arbitrarily large (for example, the clique-width of a complete graph is at most 2 and its treewidth is the number of vertices minus 1). The known results on these two parameters so far make treewidth a much more manageable graph parameter than clique-width. Although treewidth is also NP-hard to compute in general, by Bodlaender's celebrated result, graphs of bounded treewidth can be recognised in linear time [1], and the complexity of computing the treewidth is known for most of the well-known graph classes. We can say that treewidth is well understood, whereas the same is not true for clique-width. For example, treewidth has many characterisations, via tree decompositions, partial k-trees, embeddings into chordal graphs, graph searching, and forbidden minors. When it comes to clique-width, only little is known about characterisations. Clique-width was originally defined as the smallest number of labels needed for building a graph by application of labelled graph operations. During the last two decades, only one other characterisation has been discovered [7]: For a finite set C of labels, a *C-construction* of a graph G is a sequence $(G_0, \ldots, G_r)$ of C-labelled graphs on the vertex set of G, where each graph G_i is the disjoint union of two C-labelled graphs and G_{i+1} emerges from G_i by changing labels or by adding the edges in one union graph of G_i. The size of set C bounds the clique-width of G from above [7].

In this paper, we present a new characterisation of clique-width. It is based on a rooted binary tree that iteratively partitions the vertex set into finally singleton partition sets by obeying certain adjacency conditions. The clique-width of a graph is then determined by the number of partition sets that are active at a time. Using this characterisation, we show that k-path powers with at least $(k+1)^2$ vertices have clique-width $k+2$. This solves a long-standing open problem posed in [10]. A k-path power is the k-power graph of a simple path. The only known result for classes of unbounded clique-width so far has been for square grids: the clique-width of a $k \times k$ grid is $k+1$ [10]. Note that for each positive integer k, there is only one $k \times k$ grid, whereas there are infinitely many k-path powers.

Some earlier results that emerged from the study of the computational complexity of the relative clique-width and NLC-width problems can be seen as related to our new characterisation. NLC-width is a graph parameter similar to clique-width, and the two parameters differ by a factor of at most 2 [16]. Müller and Urner gave a characterisation of NLC-width through decomposition trees

[21]. Our characterisation of clique-width is more than a direct generalisation of previous characterisation results for NLC-width. For NLC-width, the employed precise formulation defines a partition that is unique. The context of the decomposition defined at some other node of the decomposition tree is not required, as it is implicit. This is not the case for clique-width. In particular, the partition at some tree node is *not* unique. This may also explain part of the difficulties that have arisen from understanding and working with clique-width.

2 Definitions and Notation

We consider only simple finite undirected graphs. For $G = (V, E)$ a graph, $V = V(G)$ is the *vertex set* of G and $E = E(G)$ is the edge set of G. Edges are denoted as uv, where vertices u and v are *adjacent* in G, i.e., u is a *neighbour* of v in G. For a vertex u of G, the *(open) neighbourhood* of u, denoted as $N_G(u)$, is the set of neighbours of u in G. A graph H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For a set $X \subseteq V(G)$, the subgraph of G *induced* by X, denoted as $G[X]$, is the subgraph H of G such that $V(H) = X$ and for every vertex pair u, v of H, if $uv \in E(G)$ then $uv \in E(H)$. For two graphs G and H, where $V(G) \cap V(H) = \emptyset$, the *disjoint union* of G and H, denoted as $G \oplus H$, is the graph $(V(G) \cup V(H), E(G) \cup E(H))$.

For an integer $k \geq 1$, a *k-labelled graph* is an ordered triple $G = (V, E, \ell)$ where (V, E) is a graph and $\ell : V \to \{1, \ldots, k\}$, the *label function* of G. We define four types of operations for k-labelled graphs:

- for $1 \leq i \leq k$ and u a vertex, $i(u)$ is the k-labelled graph $\big(\{u\}, \emptyset, \{(u, i)\}\big)$.
- for $1 \leq i < j \leq k$ and G a k-labelled graph, $\eta_{i,j}(G)$ is the k-labelled graph that emerges from G by adding all edges between vertices with label i and vertices with label j that are not edges of G.
- for $1 \leq i, j \leq k$, $i \neq j$, and G a k-labelled graph, $\rho_{i \to j}(G)$ is the k-labelled graph that emerges from G by changing all occurrences of label i into label j.
- for G and H k-labelled graphs, $G \oplus H$ is the k-labelled graph on vertex set $V(G) \cup V(H)$ and with edge set $E(G) \cup E(H)$ and each vertex of $G \oplus H$ has the same label as in G or H.

A *k-expression* is built from the four operation types: $i(u)$ is a k-expression, for α and ω k-expressions, $\eta_{i,j}(\alpha)$, $\rho_{i \to j}(\alpha)$ and $(\alpha \oplus \omega)$ are k-expressions. Each expression defines a tree, that we also call a *clique-width tree*. By $\mathrm{val}(\alpha)$, we denote the labelled graph that is defined by α. For a graph G, an integer k and a k-expression α, we say that α is a *k-expression for G* if there is a label function ℓ for G with $(V(G), E(G), \ell) = \mathrm{val}(\alpha)$. The *clique-width* of G, denoted as $\mathrm{cwd}(G)$, is the smallest integer k such that there is a k-expression for G.

3 Supergroup Partitions Characterise Clique-Width

We aim at a characterisation of clique-width. Previously, clique-width and its variant linear clique-width were investigated by using the graph notion of a

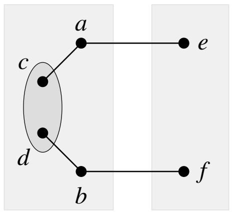


Fig. 1. The vertex set of the depicted graph, on vertices a, b, c, d, e, f, is partitioned into the sets $\{a, b, c, d\}$ and $\{e, f\}$. The sets $\{a\}, \{b\}, \{c, d\}$ are groups of $G[\{a, b, c, d\}]$.

"group". Let G be a graph and let H be an induced subgraph of G. A set $A \subseteq V(H)$ is called a *group* of H if $N_G(u) \setminus V(H) = N_G(v) \setminus V(H)$ for every vertex pair $u, v \in A$. Note the difference between this definition and the notion of a module. A set $B \subseteq V(G)$ is a *module* of G if $N_G(u) \setminus B = N_G(v) \setminus B$ for every vertex pair $u, v \in B$. Observe that B is a module of G if B is a group of $G[B]$. Since a module B of G is a group in every induced subgraph H of G that contains all vertices in B, we can say that the notion of a group is more general than the module notion.

Groups are very useful for understanding linear clique-width [11, 12, 20]. However, they are insufficient for studying clique-width. Such observations were first published by Müller and Urner in their work about the complexity of computing the relative clique-width [21]. For illustrating the situation, consider the graph, G, in Figure 1. The vertex set of the depicted graph is partitioned into $\{a, b, c, d\}$ and $\{e, f\}$, which is indicated by the two rectangle areas. The groups in the left side partition set are $\{a\}$, $\{b\}$ and $\{c, d\}$. Spending one label per group and then computing the disjoint union with $G[\{e, f\}]$ requires a label for each of e and f, and the two labels must be different from the labels used in $G[\{a, b, c, d\}]$. Hence, computing G from the disjoint union of $G[\{a, b, c, d\}]$ and $G[\{e, f\}]$ requires five labels. However, if we allow four labels in $G[\{a, b, c, d\}]$ then e and f can have the same label as respectively c and d. This requires a total of four labels for computing G from the disjoint union of $G[\{a, b, c, d\}]$ and $G[\{e, f\}]$. We will give a characterisation of clique-width by applying the ideas and observations given above by generalising the notion of a "group". Let G be a graph and let H be a subgraph of G. Denote by $G{-}H$ the graph $(V(G), E(G) \setminus E(H))$, i.e., the subgraph of G obtained from deleting the edges of H.

Definition 1. *Let G be a graph and let H be a subgraph of G.*

1) A set A of vertices of H is called group *of H if for every vertex pair u, v from A, $N_{G-H}(u) = N_{G-H}(v)$.*
2) A set A of vertices of H is called supergroup *of H if for every vertex pair u, v from A, $N_{G-H}(u) \subseteq N_G(v)$.*
3) A supergroup partition *for H is a partition $(A_1, \ldots, A_r)$ of $V(H)$ such that $A_1, \ldots, A_r$ are supergroups of H. The* size *of a supergroup partition is the number r of partition classes.*

4) *For A and B supergroups of H where $A \cap B = \emptyset$, we say that A and B are* compatible for H *if $uv \in E(G-H)$ for some $u \in A$ and $v \in B$ implies $xy \in E(G)$ for every $x \in A$ and $y \in B$.*

Note that the definition of a group in Definition 1 extends the definition of a group given in the initial paragraph of this section from induced subgraphs to arbitrary subgraphs. Formally, a supergroup is a more general notion than group. We illustrate the difference between the two notions by a simple example. Consider the graphs $G = (\{a, b, c\}, \{ab, bc\})$ and $H = (\{a, b, c\}, \{ab\})$. Note that H is a subgraph of G, that is not an induced subgraph, and that $G-H = (\{a, b, c\}, \{bc\})$. The set $\{a, c\}$ is a supergroup of H, but it is not a group of H. It is not difficult to see that the group and supergroup notion coincide on induced subgraphs. A simple but important property of supergroups is that pairs of vertices that are adjacent in $G-H$ cannot belong to the same supergroup.

We use the supergroup notion to define a recursive partition of the vertex set of graphs. Let $\mathscr{M} = (M_1, \ldots, M_r)$ and $\mathscr{N} = (N_1, \ldots, N_s)$ be partitions of a universe X. We say that $\mathscr{N}$ is a *refinement* of $\mathscr{M}$, if $N_1 \cup \cdots \cup N_s = M_1 \cup \cdots \cup M_r$ and for every $1 \leq i \leq s$, there is $1 \leq j \leq r$ such that $N_i \subseteq M_j$.

Definition 2. *Let G be a graph. A* supergroup tree *for G is a rooted binary tree T whose nodes are labelled with partitions of subsets of $V(G)$ such that the following conditions are satisfied:*

1) *for $x_1, \ldots, x_n$ the vertices of G, there is a 1-to-1 correspondence between the leaves of T and the singleton set partitions $(\{x_1\}), \ldots, (\{x_n\})$.*
2) *let a be an inner node of T with sons b and c, and let a, b, c be labelled with the partitions $(A_1, \ldots, A_p)$, $(B_1, \ldots, B_q)$ and $(C_1, \ldots, C_r)$, respectively; then,*
   - *$(B_1, \ldots, B_q, C_1, \ldots, C_r)$ is a refinement of $(A_1, \ldots, A_p)$.*
   - *$(A_1, \ldots, A_p)$ is a supergroup partition for $G[B_1 \cup \cdots \cup B_q] \oplus G[C_1 \cup \cdots \cup C_r]$.*
   - *$A_1, \ldots, A_p$ are pairwise compatible for $G[B_1 \cup \cdots \cup B_q] \oplus G[C_1 \cup \cdots \cup C_r]$.*

The size *of T is the largest size of a supergroup partition a node of T is labelled with.*

In the second condition of Definition 2, note that $G[B_1 \cup \cdots \cup B_q]$ and $G[C_1 \cup \cdots \cup C_r]$ are induced subgraphs of G but their disjoint union, $G[B_1 \cup \cdots \cup B_q] \oplus G[C_1 \cup \cdots \cup C_r]$, may not be an induced subgraph of G. This is the case, when G has an edge joining a vertex from $B_1 \cup \cdots \cup B_q$ and a vertex from $C_1 \cup \cdots \cup C_r$; such edges are not contained in $G[B_1 \cup \cdots \cup B_q] \oplus G[C_1 \cup \cdots \cup C_r]$.

Let T be a rooted tree and a a node of T. By T_a, we denote the subtree of T that is rooted at a. Let G be a graph and let α be an expression for G. Let $T = T[\alpha]$ be the clique-width tree that is defined by α. For a node a of T and a vertex x of G, we say that x *occurs* at a in T, if $i(x)$ for some label i is the label of some leaf in T_a.

Lemma 1. *There is a function that, given a graph G and a k-expression α for G, computes a supergroup tree for G of size at most k in $\mathcal{O}(n^2)$ time.*

Proof. Let G be a graph, and let α be a k-expression for G, where $k \geq 1$. We can assume that α contains only useful operations. We describe the construction of a supergroup tree for G of size at most k. Denote by $T[\alpha]$ the clique-width tree of α. Note that there is a 1-to-1 correspondence between the leaves of $T[\alpha]$ and the vertices of G. Obtain the rooted binary tree T from $T[\alpha]$ as follows: there is a 1-to-1 correspondence between the leaves of $T[\alpha]$ and the leaves of T and there is a 1-to-1 correspondence between the $\oplus$-labelled nodes of $T[\alpha]$ and the nodes of T such that for every inner node a of T, if B and C are the sets of vertices that appear in the two subtrees of T rooted at a, B and C are the sets of vertices appearing in the two subtrees of $T[\alpha]$ rooted at the image of a in $T[\alpha]$. Informally spoken, T is obtained from $T[\alpha]$ by contracting the edges that are incident to an inner node that is not labelled with $\oplus$. We add labels to the nodes of T in the following way. Let a be a node of T, and let a' be the node of $T[\alpha]$ that corresponds to a. Let X be the set of vertices that occur at a' in $T[\alpha]$ and let $(X_1, \ldots, X_r)$ be the partition of X that is defined by the labels at a'. Note that $r \leq k$. Then, label a in T with $(X_1, \ldots, X_r)$. This completes the definition of T.

It can be verified that the labelled tree T satisfies the conditions of Definition 2. Thus, T is a supergroup tree for G. Furthermore, since every partition label of T has size at most k, it follows that T is a supergroup tree for G of size at most k. The claimed running time follows from a careful analysis of the above developed algorithm. □

Lemma 2. *Let G be a graph. Let T be a supergroup tree for G of size t. Then, G has a k-expression α for some $k \leq t$.*

Proof. Let a be a node of T and with supergroup partition label $(A_1, \ldots, A_p)$; let $A =_{\text{def}} A_1 \cup \cdots \cup A_p$. We inductively construct an expression for $G[A]$, where the labels induce a partition of A that is equal to $(A_1, \ldots, A_p)$. If a is a leaf of T then $A = \{x\}$ for some vertex x of G, and $1(x)$ is an appropriate expression. Let a be an inner node of T, that has the two sons b and c. Let b and c be labelled with $(B_1, \ldots, B_q)$ and $(C_1, \ldots, C_r)$, respectively. There are appropriate expressions β and γ for respectively $G[B_1 \cup \cdots \cup B_q]$ and $G[C_1 \cup \cdots \cup C_r]$. Then, $(\beta \oplus \gamma)$ is an expression for $G[B_1 \cup \cdots \cup B_q] \oplus G[C_1 \cup \cdots \cup C_r]$. The desired expression for $G[A]$ is obtained from β and γ in two steps: first, changing labels in β and γ, and second, adding the new edges. This can be done without using more labels than the size of T. □

Lemma 1 and Lemma 2 imply

Theorem 1. *Let G be a graph. The smallest size of a supergroup tree for G is equal to the clique-width of G.*

4 The Clique-Width of Large Path Powers

We aim at developing an efficient algorithm for computing the clique-width of a class of proper interval graphs. The algorithm itself will be very simple, since

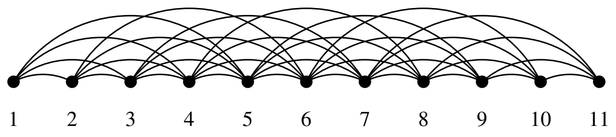


Fig. 2. The figure shows a 4-path power on eleven vertices. A pair of vertices is adjacent if and only if the difference of their name label is at most 4. For example, 2 and 10 are neighbours of 6, but 1 and 11 are non-adjacent to 6.

it will compute the clique-width from the size of a largest clique. The size of a largest clique of a proper interval graph can be computed in linear time. The main theoretical result of this section will be the correctness proof of the algorithm, precisely, we will show a tight lower bound on the clique-width of the considered class of proper interval graphs.

We begin by defining the graphs that we will consider. For $k \geq 1$, a graph G is a *k-path power* if its vertices admit an ordering $\langle v_1, \ldots, v_n \rangle$ such that v_i and v_j are adjacent if and only if $0 < |i - j| \leq k$. As an example, the 1-path powers are exactly the induced paths. A *path power* is a k-path power for some $k \geq 1$. An example of a 4-path power using the vertex ordering definition is depicted in Figure 2. Every path power is a proper interval graph. Furthermore, path powers can be recognised in linear time. The size of a largest clique of a k-path power on at least $k+1$ vertices is $k+1$. We say that a k-path power is *large* if it contains at least $(k+1)^2$ vertices. An upper bound on the clique-width of large path powers is known.

Lemma 3 ([9]). *Let G be a large k-path power, with $k \geq 1$. The clique-width of G is at most $k+2$.*

For a lower bound, it is known that the clique-width of a large k-path power is more than k [10]. This leaves a gap of 1 between the known upper and lower bounds on the clique-width of large k-path powers. In this section, we close the gap by showing that the upper bound of Lemma 3 is tight. We will obtain this result by determining a lower bound on the clique-width of large path powers of smallest size. For $n \geq 2$, a *proper interval square*, *square* for short, denoted as Q_n, is the $(n-1)$-path power on n^2 vertices. The vertices of Q_n are $v_{1,1}, \ldots, v_{n,1}, v_{1,2}, \ldots, v_{n,n}$, and the edges of Q_n are determined by vertex ordering $\langle v_{1,1}, \ldots, v_{n,1}, v_{1,2}, \ldots, v_{n,n} \rangle$ in the sense of the above definition of k-path powers. This means that for each pair i, j:

$$N_{Q_n}(v_{i,j}) = \left\{ v_{i+1,j-1}, \ldots, v_{n,j-1}, v_{1,j}, \ldots, v_{n,j}, v_{1,j+1}, \ldots, v_{i-1,j+1} \right\} \setminus \left\{ v_{i,j} \right\};$$

in border cases, some of the listed vertices may not exist, that we simply exclude in such cases. We partition the vertices of Q_n: a *column* of Q_n is a set $\{v_{1,j}, \ldots, v_{n,j}\}$. We often speak of "column j", which means exactly the vertices $v_{1,j}, \ldots, v_{n,j}$. Small examples of squares are depicted in Figure 3, where the vertices are arranged analogous to this representation.

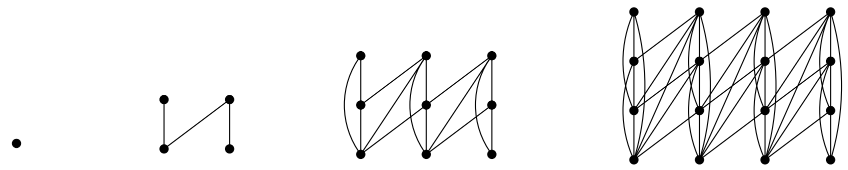


Fig. 3. Depicted are four proper interval squares, Q_1, Q_2, Q_3 and Q_4. For each graph Q_n, the upper left and lower right vertex are respectively $v_{1,1}$ and $v_{n,n}$.

For showing the lower bound on the clique-width of Q_n, we will apply Theorem 1, which means, we will determine a lower bound on the size of supergroup trees for Q_n. We first present three auxiliary results on the size of supergroup partitions for particular subgraphs of Q_n. Let $A \subseteq V(Q_n)$ and $1 \leq s \leq n$. The *s-boundary* of A is the set $\{v_{p_1,q_1}, \ldots, v_{p_r,q_r}\}$ of vertices from A such that for every $1 \leq i \leq r$, $q_i < s$ and $v_{p_i,q_i+1}, \ldots, v_{p_i,s-1} \notin A$. We say that column s is *full* in A if $\{v_{1,s}, \ldots, v_{n,s}\} \subseteq A$, and we say that column s is *empty* in A if $v_{1,s}, \ldots, v_{n,s} \notin A$.

Lemma 4 ([13]). *Let $A \subseteq V(Q_n)$, and let e be an empty column of A. The vertices of the e-boundary of A are in pairwise different supergroups of $Q_n[A]$.*

For $A \subseteq V(Q_n)$, we say that A has the *filled row* property if for all $v_{i,j}, v_{i,j'} \in A$ where $j < j'$, $\{v_{i,j}, \ldots, v_{i,j'}\} \subseteq A$. The result of the following lemma can be shown by a careful case analysis.

Lemma 5. *Let $A \subseteq V(Q_n)$. Let f be smallest such that $\{v_{1,f}, \ldots, v_{n,f}\} \subseteq A$, and let there be $e > f$ such that $v_{1,e}, \ldots, v_{n,e} \notin A$. If $|A| > n$ then A satisfies one of the two conditions:*

1) $f \leq 2$ and $v_{1,1} \in A$ and A has the filled row property.
2) every supergroup partition of $Q_n[A]$ has size more than n.

For $1 \leq i \leq n$ and $1 < j \leq n$, we call (A, B) a *partial $[i,j]$-partition* of $V(Q_n)$ if $A \subseteq \{v_{1,1}, \ldots, v_{n,j-1}\} \cup \{v_{1,j}, \ldots, v_{i-1,j}\}$ and $B \subseteq \{v_{i,j}, \ldots, v_{n,j}\} \cup \{v_{1,j+1}, \ldots, v_{n,n}\}$. For u, v, w a vertex triple of Q_n, we say that u *distinguishes* v and w if u is adjacent to exactly one of v and w. If uv is an edge of Q_n and the two vertices are non-adjacent in a subgraph then v and w cannot be contained in the same supergroup of the subgraph.

Lemma 6. *Let $n \geq 3$, and let $1 \leq i \leq n$ and $1 < j \leq n$. Let (A, B) be a partial $[i,j]$-partition of $V(Q_n)$ such that A has a full column and B is non-empty. Furthermore, let $j = n$ imply $i = 1$ and let $j = n$ and $v_{n,n} \in B$ imply $|B| \geq 2$. Let $(X_1, \ldots, X_k)$ be a supergroup partition for $Q_n[A] \oplus Q_n[B]$ where $X_1, \ldots, X_k$ are pairwise compatible for $Q_n[A] \oplus Q_n[B]$. Then, $k \geq n + 1$.*

Proof. For a contradiction, we assume that $k \leq n$. If B contains no vertex from column j then (A, B) is also a partial $[1, j+1]$-partition of $V(Q_n)$. Note that

$B \neq \emptyset$ implies $j < n$. Iterating this argument, we can henceforth assume that B contains a vertex from column j. By assumption about j, column n is empty in A. Thus, the vertices of the n-boundary of A are in pairwise different supergroups due to Lemma 4. Since there is a full column in A by our assumption, the n-boundary of A consists of n vertices. It follows that $k = n$ and each supergroup contains an n-boundary vertex of A. Let s be smallest such that $v_{s,j} \in B$. Note that (A, B) is a partial $[s, j]$-partition of $V(Q_n)$. We distinguish between $s = n$ and $s < n$.

Suppose that $s = n$. Note that our assumptions directly imply $j < n$. Let $1 \leq i \leq k$ be such that $v_{n,j} \in X_i$. Let v be the vertex from the n-boundary of A with $v \in X_i$. Remember that v exists and is unique due to the results of the first paragraph. Since $v_{n,j}$ is adjacent to all vertices from column j in Q_n, it follows that v is not from column j. Since $v_{n-1,j} \notin B$ due to assumption $s = n$, it follows that $v_{n-1,j} \notin X_i$ and thus v must be adjacent to $v_{n-1,j}$. This directly implies that $v = v_{n,j-1}$. Since $v_{1,j+1}$ would distinguish $v_{n,j}$ and v, it holds that $v_{1,j+1} \in B$. Let $1 \leq i' \leq k$ be such that $v_{1,j+1} \in X_{i'}$. Since $v_{1,j} \notin B$, $v_{n,j}$ and $v_{1,j+1}$ are distinguished by $v_{1,j}$, and thus, $i \neq i'$. Let v' be the vertex from the n-boundary of A with $v' \in X_{i'}$. Since $v_{n-1,j}$ is not in B and adjacent to $v_{1,j+1}$, it follows that v' is adjacent to $v_{n-1,j}$. And since $v_{1,j+1}$ and v' must be non-adjacent in Q_n, v' is either $v_{n,j-1}$ or $v_{1,j}$. The former case would contradict $i \neq i'$. The latter case implies that the vertices from X_i and $X_{i'}$ are pairwise adjacent in Q_n because of $v_{1,j}v_{n,j} \in E(Q_n) \setminus E(Q_n[A] \oplus Q_n[B])$, which yields a contradiction because of $v_{n,j-1}v_{1,j+1} \notin E(Q_n)$. So, $s < n$ must hold.

Let $1 \leq t \leq k$ be such that $v_{s,j} \in X_t$. Suppose that $v_{n,j} \notin B$. Then, $v_{s,j}$ is distinguished by $v_{n,j}$ from all vertices $v_{1,1}, \ldots, v_{n,j-1}$, which means that X_t does not contain any vertex from the columns $1, \ldots, j-1$. And since $v_{s,j}$ is adjacent to every other vertex from column j, X_t cannot contain any vertex from column j that is in A. Thus, X_t does not contain any vertex from A. In particular, X_t does not contain any vertex from the n-boundary of A, which contradicts the assumptions from the first paragraph. We conclude $v_{n,j} \in B$. Let $1 \leq p \leq k$ be such that $v_{n,j} \in X_p$. Since $v_{s,j}$ is distinguished from $v_{n,j}$ by $v_{n,j-1}$, it follows that $p \neq t$. Let v_p and v_t be the vertices from the n-boundary of A that are contained in X_p and X_t, respectively. Since $v_t \in A$ and $v_{s,j} \in B$ and v_t and $v_{s,j}$ are in the same supergroup X_t, it follows that v_t and $v_{s,j}$ are non-adjacent in Q_n. Consider $v_{s-1,j}$ or $v_{n,j-1}$, depending on whether $s \geq 2$ or $s = 1$. Since $v_{s-1,j} \notin B$ in case of $s > 1$ or $v_{n,j-1} \notin B$ in case of $s = 1$, and in each case the considered vertex is adjacent to $v_{s,j}$ in Q_n, it follows from the definition of supergroup that the vertex is adjacent also to v_t, and since v_t is non-adjacent to $v_{s,j}$, we conclude that $v_t = v_{s,j-1}$. We distinguish between $s > 1$ and $s = 1$. Suppose $s > 1$. If v_p is adjacent to $v_{s,j}$ then, due to X_p and X_t being compatible, the vertices in X_t and X_p are pairwise adjacent, in particular, $v_{n,j}$ is adjacent to $v_{s,j-1}$. This yields a contradiction to the definition of Q_n. Thus, v_p is non-adjacent to $v_{s,j}$. Since $v_{s-1,j} \notin B$, since $v_{n,j}$ and $v_{s-1,j}$ are adjacent in Q_n and since v_p and $v_{n,j}$ are non-adjacent in Q_n, it holds that $v_p \in \{v_{s+1,j-1}, \ldots, v_{n,j-1}\}$. This, however, would mean that v_p is adjacent to $v_{s,j}$ in Q_n, a contradiction. Hence, $s = 1$, which means that $v_t = v_{1,j-1}$ and $v_{s,j} = v_{1,j}$. If there is $1 < i < n$ such that $v_{i,j} \notin B$, then $v_{i,j}$ distinguishes $v_{1,j-1}$ and $v_{1,j}$. Thus, $\{v_{1,j}, \ldots, v_{n,j}\} \subseteq B$. Consider $v_{2,j}$. Note that $v_{1,j}$ and $v_{2,j}$ are distinguished by $v_{2,j-1}$ and that $v_{2,j-1} \notin B$. So, $v_{2,j} \notin X_t$. Let $1 \leq q \leq k$ be such that $v_{2,j} \in X_q$. Let v_q be the vertex from the n-boundary of A which is contained in X_q. Since v_q is non-adjacent to $v_{2,j}$ due to being in the same supergroup and is adjacent to $v_{n,j-1}$, it follows that $v_q = v_{2,j-1}$. It remains to observe that $v_{2,j-1}$ and $v_{1,j}$ are adjacent in Q_n and that $v_{1,j-1}$ and $v_{2,j}$ are non-adjacent in Q_n, which yields a contradiction. We conclude that $k \leq n$ is not possible, and thus $k \geq n+1$. □

We prove lower bound results for the size of special supergroup partitions of $V(Q_n)$. These results are applied in the lower bound proof of Lemma 10. Let T be a supergroup tree. Let a be a node of T, and let $(A_1, \ldots, A_r)$ be the supergroup partition that a is labelled with in T. By M_a^T, we denote the union $A_1 \cup \cdots \cup A_r$, which is the set of vertices that occur in T_a. Since the context T will always be clear, we will usually omit the superscript and simply write M_a.

Lemma 7. *Let $n \geq 3$, and let T be a supergroup tree for Q_n. Assume that T has an inner node a with b and c its sons such that $M_b = \{v_{1,f}, \ldots, v_{n,f}\}$ for some $1 \leq f \leq n$, there is no full column in M_c and there is no empty column in M_a. Then, the size of T is at least $n+1$.*

Proof. Let $(A_1, \ldots, A_r)$ be the supergroup partition that a is labelled with in T. For every pair p, p' where $1 \leq p < p' \leq n$, $v_{p,f}$ and $v_{p',f}$ are distinguished by $v_{p,f+1}$ or $v_{p',f-1}$, depending on whether $f \leq n-1$ or $f \geq 2$. Thus, the vertices from M_b appear in n pairwise different supergroups of $(A_1, \ldots, A_r)$. Assume that $f \leq n-1$. Let $1 \leq p \leq n$ be smallest such that $v_{p,n} \in M_c$. Note that p exists, since column n is not empty in M_a, and $v_{1,n}, \ldots, v_{p-1,n} \notin M_a$. Let $1 \leq i \leq r$ be such that $v_{p,n} \in A_i$. If A_i contains no vertex from M_b then $r \geq n+1$. Otherwise, A_i contains vertices from M_b; let $1 \leq q \leq n$ be such that $v_{q,f} \in A_i$. If $|n-f| \geq 2$, i.e., if $f \leq n-2$, then column n must be full in M_a, which implies that M_c contains a full column and thus contradicts the assumptions. Hence, $f = n-1$. Since $v_{q,f}$ and $v_{p,n}$ are not adjacent, $q \leq p$. If $q < n$ then $v_{n,n-2}$ distinguishes $v_{q,f}$ and $v_{p,n}$, which yields a contradiction. So, $q = n$, and thus, $p = n$. Due to the choice of p, it follows that $\{v_{1,n-1}, \ldots, v_{n,n-1}, v_{n,n}\} \subseteq M_a$ and $v_{1,n}, \ldots, v_{n-1,n} \notin M_a$. A careful analysis shows that this situation implies the claimed lower bound on the size of T. If $f = n$ then there is $1 \leq p \leq n$ such that $v_{p,1} \in M_c$. Due to our assumptions about M_a, there is $1 \leq p' \leq n$ such that $v_{p',1} \notin M_a$ and therefore distinguishes $v_{p,1}$ and every vertex from column f. Thus, there is no $1 \leq i \leq r$ such that A_i contains $v_{p,1}$ and a vertex from column f, so that $r \geq n+1$. □

Lemma 8. *Let $n \geq 3$, and let T be a supergroup tree for Q_n. Assume that T has an inner node a with b and c its sons such that M_a has a full column, M_a has no empty column, M_b and M_c have no full columns. Then, the size of T is at least $n+1$.*

Proof. Let $(A_1, \ldots, A_r)$ be the supergroup partition that a is labelled with in T. Denote by U_b the set of vertices from M_b that are "highest" in their column. Formally, $v_{i,j} \in U_b$ if $v_{i,j} \in M_b$ and for every $1 \leq i' < i$, $v_{i',j} \notin M_b$. Analogously, define U_c. We first show that vertices from U_b and U_c cannot appear in the same supergroup. So, for a contradiction, assume that there are $1 \leq i \leq r$ and $v_{p,q} \in U_b$ and $v_{p',q'} \in U_c$ such that $v_{p,q}, v_{p',q'} \in A_i$. Without loss of generality, we may assume $q \leq q'$. Since $v_{p,q}$ and $v_{p',q'}$ must be non-adjacent in Q_n, it directly follows that $q < q'$. Then, the properties of supergroups imply that $\{v_{1,q}, \ldots, v_{p-1,q}\} \subseteq M_b$. The definition of U_b therefore implies $p = 1$. Since $v_{1,q}$ is non-adjacent to every vertex from column q', column q' must be full in M_c due to the properties of vertices in the same supergroup, which contradicts the assumptions about c. Thus, no vertex from U_b appears in the same supergroup as a vertex from U_c. Next, we show that the vertices from U_b and from U_c appear in pairwise different supergroups. By analogy, it suffices to show the result for U_b. Let $v = v_{p,q}$ and $v' = v_{p',q'}$ be (different) vertices from U_b, where we can assume without loss of generality that $q < q'$. Remember that $q = q'$ is not possible due to the definition of U_b. If $p > 1$ then $v_{1,q} \notin M_b$ and therefore distinguishes v and v'. Analogously, if $p = 1$ and $p' > 1$ then $v_{1,q'} \notin M_b$ and distinguishes v and v'. Assume that $p = p' = 1$. Then, there exists a vertex $w = v_{i,q'}$ such that $w \notin M_b$, since M_b has no full column. Observe that w distinguishes v and v'. We conclude that v and v' are not contained in the same supergroup. It follows that the vertices from $U_b \cup U_c$ appear in pairwise different supergroups, which implies $|U_b| + |U_c| \leq r$. Due to the assumptions about M_a, M_b, M_c, particularly since M_a has a full and no empty column, we conclude that $|U_b| + |U_c| \geq n + 1$. $\square$

Lemma 9. *Let $n \geq 3$, and let T be a supergroup tree for Q_n. Assume that T has an inner node a with b and c its sons such that M_b has a full column and an empty column and $|M_b| \geq n + 1$. Then, the size of T is at least $n + 1$.*

Proof. By a symmetry argument, we can assume that there are $1 \leq f < e \leq n$ such that column f is full in M_b and column e is empty in M_b and column j is not full for every $1 \leq j < f$ and column j is not empty for every $f < j < e$. For a contradiction, suppose that the size of T is at most n. We apply Lemma 5 and directly conclude that $f \leq 2$ and $v_{1,1} \in M_b$ and M_b has the filled row property. Let $(A_1, \ldots, A_r)$ be the supergroup partition that a is labelled with in T. Consider the e-boundary of M_b; it contains n vertices. We apply Lemma 4 and see that the e-boundary vertices of M_b appear in pairwise different supergroups of the partition. Due to the assumption, it follows that every A_i contains exactly one e-boundary vertex of M_b. We show that (M_b, M_c) is a partial $[i,j]$-partition of $V(Q_n)$ for appropriate i, j. Let $1 \leq q \leq n$ be smallest such that there is $1 \leq p \leq n$ with $v_{p,q} \in M_c$; we choose p smallest possible. Remember that M_c is non-empty. Let $1 \leq l \leq r$ be such that $v_{p,q} \in A_l$. If $q = 1$ then $f = 2$, and since $v_{p,q}$ is adjacent to $v_{1,1}$ and no vertex from the e-boundary of M_b is adjacent to $v_{1,1}$, A_l cannot contain an e-boundary vertex of M_b, a contradiction. So, let $q \geq 2$, which means $q > f$. If $q \geq e$ then (M_b, M_c) is a partial $[1, q]$-partition of $V(Q_n)$. This directly follows from the filled row property of M_b. Assume that

A_l contains an e-boundary vertex $v_{p',q'}$ of M_b. Let $q < e$. If $q < q'$ then, due to the filled row property of M_b, $v_{p,q'} \notin M_b$, which yields a contradiction to A_l being a supergroup of $Q_n[M_b] \oplus Q_n[M_c]$. So, $q' < q$. If $q' \leq q - 2$ then $\{v_{1,q}, \ldots, v_{n,q}\} \subseteq M_c$, so that column q is empty in M_b. Since $f < q < e$, we obtain a contradiction to the assumption about e. Thus, $q' = q - 1$, and since $v_{p,q}$ and $v_{p',q'}$ are non-adjacent in Q_n, $p' \leq p$ due to the adjacency definitions. It follows that $\{v_{p,q}, \ldots, v_{n,q}, v_{1,q+1}, \ldots, v_{p-1,q+1}\} \subseteq M_c$. Since M_b has the filled row property, we conclude that (M_b, M_c) must be a partial $[p, q]$-partition of $V(Q_n)$.

So, we have seen that (M_b, M_c) is a partial $[i, j]$-partition of $V(Q_n)$, and M_b has a full column and M_c is non-empty. If (M_b, M_c) is a partial $[i, j]$-partition for some $1 < j < n$ then Lemma 6 implies $r \geq n + 1$, and so the claim follows. Otherwise, (M_b, M_c) is a partial $[1, n]$-partition, and $e = n$. If $\{v_{1,n}, \ldots, v_{n-1,n}\} \cap M_c \neq \emptyset$ then, again, we apply Lemma 6 and conclude $r \geq n + 1$. Otherwise, $M_c = \{v_{n,n}\}$, which implies that $p = n$ and $q = n$. Since the neighbours of $v_{n,n}$ in Q_n are exactly $v_{1,n}, \ldots, v_{n-1,n}$, $q' = n - 1$ and $\{v_{1,n-1}, \ldots, v_{n,n-1}\} \subseteq M_b$. It follows that $\{v_{1,n-1}, \ldots, v_{n,n-1}, v_{n,n}\} \subseteq M_a$ and $v_{1,n}, \ldots, v_{n-1,n} \notin M_a$, and by carefully analysing the situation, it can be shown that the size of T is at least $n + 1$. □

Now, we are ready to complete the lower bound proof. We particularly show that one of the situations in the already given lemmas must occur in a supergroup tree for Q_n.

Lemma 10. *For every $n \geq 3$, $\mathrm{cwd}(Q_n) \geq n + 1$.*

Proof. Let $n \geq 3$, and let T be an arbitrary supergroup tree for Q_n. Let F be the set of nodes x of T such that M_x has a full column. We call a node in F *minimal* if its sons do not belong to F. Assume that there is a minimal node a in F such that M_a has no empty column. Then, a and its two sons satisfy the conditions of Lemma 8, and we conclude that the size of T is at least $n + 1$. Assume that for every minimal node x in F, M_x has an empty column. Let a be an inner node of T with its sons b and c such that b is a minimal node in F. Then, M_b has a full and an empty column. If we can choose a, b, c so that $|M_b| \geq n + 1$ then the three nodes satisfy the conditions of Lemma 9, and we conclude that the size of T is at least $n + 1$. If a, b, c cannot be chosen to satisfy the conditions of Lemma 9 then $|M_x| = n$ for every minimal node in F. So, let a, b, c be an arbitrary choice such that b and c are the sons of a and b is a minimal node from F. Let a' be the parent of a in T, if it exists. If c is a minimal node from F then $|M_a| = |M_b| + |M_c| = 2n$ and M_a has a full and an empty column, and therefore, a' satisfies the conditions of Lemma 9. Otherwise, if c is not a minimal node from F, then T_c may or may not contain a node from F. If the former then, due to the assumptions about the choice of c as not being minimal, $|M_c| \geq n + 1$, $|M_c|$ contains a full column and, since M_b contains a full column, M_c contains an empty column. Thus, node a satisfies the conditions of Lemma 9. If the latter then, if M_a contains no empty column, the nodes a, b, c

satisfy the conditions of Lemma 7, if M_a contains an empty column, a' satisfies the conditions of Lemma 9. The claim of the lemma follows by application of Theorem 1. □

Theorem 2. *Let* $k \geq 1$*, and let* G *be a large* k*-path power. Then,* $\text{cwd}(G) = k+2$*.*

Proof. Due to Lemma 3, $\text{cwd}(G) \leq k + 2$. For the lower bound, note that G contains Q_{k+1} as induced subgraph. If $k \geq 2$ then $\text{cwd}(G) \geq \text{cwd}(Q_{k+1}) \geq k+2$ due to Lemma 10, if $k = 1$ then Q_{k+1} is an induced path of length 3, and thus $\text{cwd}(G) \geq \text{cwd}(Q_{k+1}) \geq k + 2 = 3$. □

As a corollary, by applying the same algorithm as in [13], the result of Theorem 2 directly implies a linear-time algorithm for computing the clique-width of large path powers.

5 Final Remarks

We have shown two main results. The first main result is a purely graph-theoretic characterisation of clique-width by using partition trees. We believe that this provides a new view on clique-width and may lead to interesting theoretic and algorithmic results.

The second main result is the characterisation of the clique-width of a class of proper interval graphs. Except for the class of square grids, no other graph class of unbounded clique-width is known for which such a characterisation result exists. The main technical results for achieving the characterisation provided lower bounds for particular subgraphs and showed in the proof of Lemma 10 that it suffices to consider only such subgraphs. The proof of Lemma 10 is also interesting from a broader perspective. All arguments considered only very local properties. It was sufficient to determine a lower bound on the size of supergroup partitions for subgraphs of the type $Q_n[A] \oplus Q_n[B]$, independent of constraints that would be imposed by situations at other nodes of the supergroup tree. It seems that such a property makes it comparably "easy" to analyse the clique-width. Is there a general scheme behind? Is this true for graphs of specific structural properties?

Theorem 2 together with the results in [13] shows that clique-width and its variant linear clique-width coincide on large path powers. We obtained this result by explicitly proving the bounds. Can this result be shown also by using only structural arguments?

References

1. Bodlaender, H.: A Linear-Time Algorithm for Finding Tree-Decompositions of Small Treewidth. SIAM Journal on Computing 25, 1305–1317 (1996)
2. Brandstädt, A., Dragan, F., Le, H.-O., Mosca, R.: New Graph Classes of Bounded Clique-Width. Theory of Computing Systems 38, 623–645 (2005)
3. Corneil, D.G., Habib, M., Lanlignel, J.-M., Reed, B.A., Rotics, U.: Polynomial time recognition of clique-width ≤ 3 graphs. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 126–134. Springer, Heidelberg (2000)

4. Corneil, D.G., Rotics, U.: On the Relationship between Clique-width and Treewidth. SIAM Journal on Computing 34, 825–847 (2005)
5. Courcelle, B., Engelfriet, J., Rozenberg, G.: Handle-rewriting hypergraph grammars. Journal of Computer and System Sciences 46, 218–270 (1993)
6. Courcelle, B., Makowsky, J.A., Rotics, U.: Linear time solvable optimization problems on graphs of bounded clique-width. Theory of Computing Systems 33, 125–150 (2000)
7. Courcelle, B., Olariu, S.: Upper bounds to the clique width of graphs. Discrete Applied Mathematics 101, 77–114 (2000)
8. Espelage, W., Gurski, F., Wanke, E.: Deciding clique-width for graphs of bounded tree-width. Journal of Graph Algorithms and Applications 7, 141–180 (2003)
9. Fellows, M.R., Rosamond, F.A., Rotics, U., Szeider, S.: Clique-Width is NP-Complete. SIAM Journal on Discrete Mathematics 23, 909–939 (2009)
10. Golumbic, M.C., Rotics, U.: On the Clique-Width of Some Perfect Graph Classes. International Journal of Foundations of Computer Science 11, 423–443 (2000)
11. Gurski, F.: Linear layouts measuring neighbourhoods in graphs. Discrete Mathematics 306, 1637–1650 (2006)
12. Heggernes, P., Meister, D., Papadopoulos, C.: Graphs of small bounded linear clique-width. Technical report 362 in Informatics, University of Bergen (2007)
13. Heggernes, P., Meister, D., Papadopoulos, C.: A Complete Characterisation of the Linear Clique-Width of Path Powers. In: Chen, J., Cooper, S.B. (eds.) TAMC 2009. LNCS, vol. 5532, pp. 241–250. Springer, Heidelberg (2009)
14. Hlinený, P., Oum, S.-I., Seese, D., Gottlob, G.: Width parameters beyond tree-widthand their applications. The Computer Journal 51, 326–362 (2008)
15. Kamiński, M., Lozin, V.V., Milanič, M.: Recent developments on graphs of bounded clique-width. Discrete Applied Mathematics 157, 2747–2761 (2009)
16. Johansson, Ö.: Clique-decomposition, NLC-decomposition, and modular decomposition – relationships and results for random graphs. Congressus Numerantium 132, 39–60 (1998)
17. Lozin, V.: From tree-width to clique-width: Excluding a unit interval graph. In: Hong, S.-H., Nagamochi, H., Fukunaga, T. (eds.) ISAAC 2008. LNCS, vol. 5369, pp. 871–882. Springer, Heidelberg (2008)
18. Lozin, V., Rautenbach, D.: Chordal bipartite graphs of bounded tree- and clique-width. Discrete Mathematics 283, 151–158 (2004)
19. Lozin, V., Rautenbach, D.: On the Band-, Tree-, and Clique-Width of Graphs with Bounded Vertex Degree. SIAM Journal on Discrete Mathematics 18, 195–206 (2004)
20. Lozin, V., Rautenbach, D.: The relative clique-width of a graph. Journal of Combinatorial Theory, Series B 97, 846–858 (2007)
21. Müller, H., Urner, R.: On a disparity between relative cliquewidth and relative NLC-width. Discrete Applied Mathematics 158, 828–840 (2010)
22. Oum, S.-i., Seymour, P.: Approximating clique-width and branch-width. Journal of Combinatorial Theory, Series B 96, 514–528 (2006)

An Extended Tree-Width Notion for Directed Graphs Related to the Computation of Permanents

Klaus Meer

Lehrstuhl Theoretische Informatik
BTU Cottbus, Konrad-Wachsmann-Allee 1
03044 Cottbus, Germany
meer@informatik.tu-cottbus.de

Abstract. It is well known that permanents of matrices of bounded tree-width are efficiently computable. Here, the tree-width of a square matrix $M = (m_{ij})$ with entries from a field $\mathbb{K}$ is the tree-width of the underlying graph G_M having an edge (i,j) if and only if the entry $m_{ij} \neq 0$. Though G_M is directed this does not influence the tree-width definition. Thus, it does not reflect the lacking symmetry when $m_{ij} \neq 0$ but $m_{ji} = 0$. The latter however might have impact on the computation of the permanent. In this paper we introduce and study an extended notion of tree-width called triangular tree-width. We give examples where the latter parameter is bounded whereas the former is not. As main result we show that permanents of matrices of bounded triangular tree-width are efficiently computable. This result holds as well for the Hamiltonian Cycle problem.

1 Introduction

It is well known that the permanent of a square matrix M is hard to compute unless some major complexity theoretic conjectures fail to be true. Valiant [11] has shown that for 0-1 matrices the problem is #P-complete. And if M has entries from a field $\mathbb{K}$ of characteristic different from 2 the permanent polynomials build a VNP-complete family in Valiant's algebraic theory of complexity for families of polynomials, see [11,12] for these results. It is then natural to ask for suitable subclasses of matrices on which the computation becomes easy. Barvinok [2] has shown that families of matrices of bounded rank provide such an example. Another much more intensively studied subclass is related to the notion of tree-width of a graph (for a precise definition of tree-width see below). To each $n \times n$ matrix $M = (m_{ij})$ there naturally corresponds a directed graph $G_M = (V,E)$ on n vertices which has an edge (i,j) if and only if $m_{ij} \neq 0$. If $\{M_i\}_{i \in I}$ is a class of matrices such that the tree-width of each G_{M_i} is at most k for some fixed $k \in \mathbb{N}$ it was shown in [6] that the permanent of each of these matrices is computable in linear time in the size of the input matrix (and exponential time with respect to k). The problem thus is fixed parameter tractable with respect to the tree-width as parameter. This class of matrices and their permanent polynomials were studied further in [8,5] with respect to more precisely determining their expressive power.

One drawback of the tree-width approach with respect to permanents is that though G_M is a directed graph for the tree-width notion it has no influence whether an edge occurs in both directions or only in one. For the permanent computation, however, it might have significant impact whether one of the two entries m_{ij} and m_{ji} is non-zero or both. The goal of the present paper is to introduce and study an extended notion of tree-width called triangular tree-width which better reflects such lacking symmetry in the non-zero structure of a matrix. Starting point for the introduction of the new notion is the definition of the permanent through cycle covers of G_M. Suppose for the moment the vertices $V : \{1, \dots, n\}$ to be linearly ordered, say $1 < 2 < \dots < n$. If a cycle cover (or any cycle) contains an increasing edge with respect to this order, i.e., an (i, j) with $i < j$ it is clear that there has to be as well a decreasing edge $(\tilde{j}, \tilde{i}), \tilde{j} > \tilde{i}$ included somewhere in the cover. Thus, even though the classical tree-width of G_M might be large the more important question with respect to permanent computations is whether the two graphs given by the increasing and the decreasing edges have a better tree-width structure. Since this argument is independent of the used ordering of the vertex set we can try to look for a permutation of V that induces an optimal structure as far as the tree-widths of the two mentioned graphs are concerned. Note that with such an approach edges (i, j) and (j, i) are treated separately since always one will be decreasing and the other increasing. The minimal tree-width obtained by choosing an optimal ordering is then called the triangular tree-width of M. We show that the notion extends the classical tree-width notion on symmetric matrices. As main result we prove that the permanent of an $n \times n$-matrix is efficiently computable on classes of matrices of bounded triangular tree-width. One important feature of the corresponding algorithm is that it works in parallel with two tree-decompositions, one for the graph of increasing, the other for the graph of decreasing edges. The main difficulty for proving the main result is to analyze how the information given by the two tree-decompositions can be mixed efficiently.

The paper is structured as follows. Section 2 recalls basic notions and introduces triangular tree-width. We give an example showing that this notion extends the tree-width notion, i.e., there are families of matrices for which the former parameter is bounded whereas the latter is not. In the third section our main result is proven. It is shown both for computation of the permanent and for the Hamiltonian Cycle problem. A discussion of open questions closes the paper.

2 Basic Notions; Triangular Tree-Width

We start by recalling some basic definitions needed further on.

Definition 1. *An* arithmetic circuit *over a field* $\mathbb{K}$ *is a directed, acyclic graph; its nodes are also called* gates*. Nodes have in-degree* 0 *or* 2*, where the former* *are also the* input *nodes. There is precisely one node of out-degree 0, the* output *of the circuit. Each gate with in-degree 2 is labelled either by* $+, -$ *or* $\times$ *representing the corresponding operation in* $\mathbb{K}$*. The* size *of a circuit is its number of gates, the* depth *of the circuit is the length of the longest path from the output node to an input node.*

Circuits are used in the proof of our main theorem. The next notion is the basic one for the rest of this paper.

Definition 2. *Let* $G = (V, E)$ *be a directed graph. A tree-decomposition of* G *of width* $k \in \mathbb{N}$ *is a tree* $T = (V_T, E_T)$ *such that the conditions below are satisfied:*

(i) *to each node* $t \in V_T$ *there corresponds a subset* $X_t \subset V$ *of size at most* $k + 1$*;*
(ii) *for each edge* $(i, j) \in E$ *there is a* $t \in V_T$ *such that* $\{i, j\} \subseteq X_t$*;*
(iii) *for* $i \in V$ *let* $T(i)$ *denote those nodes* $t \in V_T$ *such that* $i \in X_t$*. Then* $T(i)$ *induces a (connected) subtree of* T*.*

Below we call X_t *also the* box *of vertices related to node* t *of* T*. The* tree-width *twd*(G) *of* G *is the minimal* k *such that* G *has a tree-decomposition of width* k*.*

The graph G can have loops but they do not influence the tree-width. Note that for the tree-width definition it is of no concern that G is directed; whenever one of the two edges (i, j) or (j, i) is present the condition to find i, j in a common box X_t applies. This is a crucial point with respect to the extended notion we are going to introduce below.

We consider $n \times n$ matrices M with entries from an underlying field $\mathbb{K}$. In the Turing model $\mathbb{K} = \mathbb{Q}$ but our results hold as well if, for example, we choose $\mathbb{K}$ to be of characteristic 0 and work in an algebraic model of computation.

To M there is attached a directed weighted graph $G_M = (V, E)$. It has $V := \{1, \ldots, n\}$ as vertex set and contains edges (i, j) if and only if $m_{ij} \neq 0$. The weight of such an edge is given as the entry $m_{ij} \in \mathbb{K}$. The permanent of M is defined using cycle covers of G_M:

Definition 3. *a)* *A* cycle cover *of a directed graph* $G = (V, E)$ *is a subset of its edges forming disjoint directed cycles such that each vertex* $i \in V$ *is incident with exactly one of the edges as outgoing edge and with exactly one as ingoing edge. Loops are allowed here. We identify each such cycle cover with the unique permutation* $\sigma \in S_n$ *of* V *describing it in the usual way.*

b) *The* weight *of a cycle cover given by permutation* σ *is the product of all weights of the edges participating in the cover, i.e.,* $\prod\limits_{i=1}^{n} m_{i\sigma(i)}$*.*

c) *A* partial cycle cover *is a subset of edges of a cycle cover. Its weight is defined accordingly.*

d) *The* permanent *of* M *is given as* $perm(M) = \sum\limits_{\sigma \in S_n} \prod\limits_{i=1}^{n} m_{i\sigma(i)}$*. Here,* S_n *denotes the set of permutations of* n *elements.*

The tree-width of a matrix M now simply is the tree-width of the attached graph G_M. As already mentioned above it does not reduce if instead of two non-zero entries m_{ij} and m_{ji} only one of the two is non-zero. However, as far as the computation of the permanent is concerned this is not at all true. An example which gives a good idea (though it finally will not fit completely into our framework, see Remark 1 below) is an upper triangular matrix M. Here the permanent just is the product of its diagonal entries. Its underlying graph however with respect to its tree-width behaves the same as the graph attached to an everywhere non-zero matrix. And the permanent of the latter of course is supposed to be hard to compute. Thus the tree-width notion seems not to reflect appropriately cases where permanents are easy to compute due to a kind of lacking symmetry in M. Triangular tree-width to be introduced now tries to capture better such situations. The basic underlying idea is to split G_M into two graphs, one consisting of increasing and the other of decreasing edges only. The words increasing and decreasing here reflect in terms of M non-zero entries occurring in the lower triangular and the upper triangular part, respectively. It is formalized using suitable permutations of rows and columns from M.

Definition 4. *Let M be an $n \times n$ matrix, $G_M = (V, E)$ its weighted directed graph with $V = \{1, \ldots, n\}$.*

a) Let $\sigma : V \to V$ be a permutation of V. The graph $G_\sigma^{inc} = (V, E_\sigma^{inc})$ is given by defining E_σ^{inc} as all edges $(i,j) \in E$ such that $\sigma(i) < \sigma(j)$. We call E_σ^{inc} the increasing *edges with respect to σ. Accordingly, $G_\sigma^{dec} = (V, E_\sigma^{dec})$ with $(i,j) \in E_\sigma^{dec}$ if and only if $(i,j) \in E$ and $\sigma(i) > \sigma(j)$ is the graph of* decreasing *edges with respect to σ.*

Without loss of generality loops are allocated to E_σ^{dec}.

b) A graph G is said to have a triangular *tree-decomposition of width $k \in \mathbb{N}$ if there is a permutation σ of its vertices such that both graphs G_σ^{inc} and G_σ^{inc} are of (normal) tree-width at most k. The* triangular tree-width *of G, in terms $ttw(G)$, is thus given as*

$$ttw(G) := \min_{\sigma \in S_n} \max\{twd(G_\sigma^{inc}), twd(G_\sigma^{dec})\} \ .$$

The ttw notion extends the normal tree-width notion in the following sense. If M is a square symmetric matrix (or, more precisely, a matrix with a symmetric non-zero structure), then $ttw(G_M) = twd(G_M)$. This is true because in that case the product $P^{-1} \cdot M \cdot P$, where P is a permutation matrix, again results in a symmetric matrix. For different permutations the graphs G^{inc} are just isomorphic copies of each other. Therefore, for each permutation σ it follows $twd(G_{M,\sigma}^{inc}) = twd(G_{M,\sigma}^{dec}) = twd(G_M)$ and the triangular tree-width of G_M coincides with the normal tree-width.

Another consequence of this remark is

Corollary 1. *Computation of the triangular tree-width of a directed graph is NP-hard.*

This follows from the corresponding result for traditional tree-width [1] and the previous remark.

2.1 A Guiding Example

In this subsection we want to study a bit more extensively an example which shows on the one hand side that our new notion captures cases which the traditional tree-width notion does not. More precisely, we consider a family of directed graphs whose tree-widths grow to infinity with the number of vertices in the graphs but whose triangular tree-widths stay bounded. On the other hand the example shows as well that the chosen permutation matters. For a not carefully chosen σ the maximum of $twd(G_\sigma^{inc})$ and $twd(G_\sigma^{dec})$ might stay unbounded. The example is a special family of grid graphs. This family is more exhaustively studied in [10] where it is also shown that its tree-width is unbounded.

For $n \in \mathbb{N}$ define a grid graph G_n with n^2 many vertices $V := \{v_{ij}, 1 \leq i, j \leq n\}$ as follows: For every $1 \leq i \leq n$ the graph G_n contains as edges the horizontal path P_i given by $v_{i1} \to v_{i2} \to \ldots \to v_{in}$. Next, G_n has two vertical paths $P'_1 : v_{11} \to v_{21} \to \ldots \to v_{n1}$ and $P'_2 : v_{1n} \to v_{2n} \to \ldots \to v_{nn}$. Finally, there are vertical edges $v_{ij} \to v_{i+1,j}$ for every pair (i, j) such that $i + j$ is even. The figure shows as example the graph G_6.

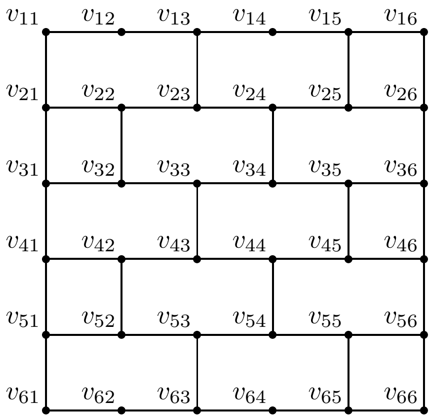

v_{11} v_{12} v_{13} v_{14} v_{15} v_{16}
v_{21} v_{22} v_{23} v_{24} v_{25} v_{26}
v_{31} v_{32} v_{33} v_{34} v_{35} v_{36}
v_{41} v_{42} v_{43} v_{44} v_{45} v_{46}
v_{51} v_{52} v_{53} v_{54} v_{55} v_{56}
v_{61} v_{62} v_{63} v_{64} v_{65} v_{66}

Fig. 1. The grid graph G_6

In order to deal with directed graphs we direct the edges as indicated above from left to right and from top to bottom. As shown in [10] there is no constant bound on the tree-width of the family $\{G_n\}_n$ as n tends to infinity. Note that in our situation we would consider such a G_n as the graph attached to an $n^2 \times n^2$ matrix M.

We now give two different permutations σ_1, σ_2 of V such that the maximum of $twd(G_{n,\sigma_1}^{inc})$ and $twd(G_{n,\sigma_1}^{dec})$ still is unbounded for growing n, whereas for all $n \in \mathbb{N}$ it is $\max\{twd(G_{n,\sigma_2}^{inc}), twd(G_{n,\sigma_2}^{dec})\} = 1$. Thus the triangular tree-width of the family $\{G_n\}_n$ is bounded but its tree-width is not.

The ordering related to permutation σ_1 is simply going from left to right in each row and then row-wise downwards. Thus for all i, j, k such that $j < k$ we put $v_{ij} <_{\sigma_1} v_{ik}$ and for all i, j, k, ℓ such that $i < j$ we put $v_{ik} <_{\sigma_1} v_{j\ell}$. Clearly,

G^{inc}_{n,σ_1} is precisely G_n since the increasing edges with respect to σ_1 correspond to the directed edges of G_n. Consequently, $twd(G^{inc}_{n,\sigma_1}) = twd(G_n)$ and therefore is unbounded for growing n.

The second permutation σ_2 is defined to be the one inducing the following linear order on the vertices. In each row we still go from left to right, but now the ordering goes row-wise from the bottom to the top. Thus for all i, j, k such that $j < k$ we have $v_{ij} <_{\sigma_2} v_{ik}$ and for all i, j, k, ℓ such that $i < j$ we put $v_{ik} >_{\sigma_2} v_{j\ell}$. Now G^{inc}_{n,σ_2} has the horizontal edges as edge set. They build n disjoint paths, so $twd(G^{inc}_{n,\sigma_2}) = 1$. The graph G^{dec}_{n,σ_2} contains the vertical edges which are all isolated, i.e., no two of them share a common vertex. Therefore $twd(G^{dec}_{n,\sigma_2}) = 1$ and $ttw(G_n) = 1$ for all $n \in \mathbb{N}$.

3 Computing Permanents of Matrices of Bounded ttw

We shall now study in how far boundedness of the triangular tree-width of the graph attached to a square matrix has impact on computing the matrix' permanent efficiently. Our results will extend the class of matrices for which efficient algorithms for this problem are known.

3.1 Particular Case: Perfect Triangular Decompositions

We shall first prove our result on a further reduced class of matrices of bounded triangular tree-width. In the next subsection the result then is shown for all such matrices.

The following technical condition on a triangular tree-decomposition of a graph will turn out to be algorithmically important below.

Definition 5. *Let $G = (V, E)$ be a directed graph, $k \in \mathbb{N}$ be fixed. G is said to have a* perfect *triangular tree-decomposition of width k if there is a permutation σ of V and two corresponding tree-decompositions $T^{inc}_\sigma, T^{dec}_\sigma$ of width k for $G^{inc}_\sigma, G^{dec}_\sigma$, respectively, such that for at least one of the two decompositions none of the vertices of G occurs in more than $10k$ many boxes. The pair of decompositions in that case is called* perfect *as well.*

The factor 10 in the definition is arbitrarily chosen. It is only important that the number of boxes in which a vertex maximally occurs is bounded in a function depending on k. Perfect decompositions are used below in order to perform a typical tree-climbing algorithm working in parallel on two tree-decompositions. Perfectness then allows to bound the resources needed by such an algorithm.

Theorem 1. *Let $k \in \mathbb{N}$ be fixed, M an $n \times n$ matrix with corresponding graph G such that G has a perfect triangular tree-decomposition of width k. Suppose this decomposition to be explicitly given. Then $perm(M)$ can be computed in polynomial time with respect to n. The problem is fixed parameter tractable with respect to parameter k, i.e., in the complexity estimates k does not occur in the exponent of n.*

Proof. The proof follows the common ideas behind bounded tree-width algorithms which climb up a rooted tree-decomposition, see for example [8]. However, due to the presence of two different graphs and their decompositions bookkeeping during such an algorithm becomes more involved. A major problem here is that a vertex which in one decomposition occurs in a leaf box only in the other might occur close to the root, and in addition it might occur in linearly many boxes of the second decomposition. The question thus is for how long an algorithm has to remember the information related to this vertex. In order to keep the related computational amount bounded the perfectness condition is employed.

Let G have perfect ttw at most k and let σ be a permutation of the vertex set witnessing this. For sake of notational simplicity we suppress the dependency on σ for the rest of this proof. Let (T^{inc}, T^{dec}) be the corresponding perfect triangular tree-decompositions of width k for G^{inc} and G^{dec}. Without loss of generality we assume the following conditions on the two trees to be satisfied:

- T^{inc} and T^{dec} are binary trees of depth $O(\log n)$, see [4];
- in both trees if t_2 is a successor node of t_1, then there is at least one vertex in $X_{t_2} \setminus X_{t_1}$;
- an increasing edge between two vertices $i, j \in V$ is represented by the unique box of T^{inc} in which both nodes occur commonly and that is closest to the root; similarly for decreasing edges.

Finally, we assume that perfectness of the pair (T^{inc}, T^{dec}) results from T^{dec}, i.e., every vertex of V occurs in at most $10k$ boxes of T^{dec}. Note that the balancing procedure in [4] does not destroy the perfectness condition.

On a high level point of view the algorithm works as follows. Climbing up in T^{inc} we build partial cycle covers from the information given in the currently treated box of T^{inc}. They consist of increasing edges with respect to σ. When the algorithm moves on from two nodes ℓ, r with related boxes X_ℓ^{inc}, X_r^{inc} to their common predecessor node t with X_t^{inc} let i be a vertex from the original graph occurring in X_ℓ^{inc} but disappearing in X_t^{inc}. Then information about partial cycle covers being made of decreasing edges incident with i is computed from T^{dec} and mixed with the information already obtained. After glueing the three nodes vertex i is deleted in the boxes of T^{dec} and the process is continued. A major point for this construction to work efficiently is that T^{dec} satisfies the perfectness condition. As consequence the amount of information T^{dec} contains with respect to vertex i is not depending on n and the mixture does not increase the complexity too much. Vertex i does not influence any more the algorithm after treatment of node X_t^{inc}.

Now towards the details of the algorithm.

The central object we compute going bottom up in T^{inc} is a catalogue of partial cycle covers for graph G, compare [8]. At each node of T^{inc} it collects information about those paths and cycles in a potential cycle cover of G that only contain edges between vertices that occur in boxes related to the subtree of T^{inc} which is rooted in t. This information is complete as far as vertices $i \in V$ are concerned that do not occur in box X_t^{inc} itself or in boxes related to nodes further up in T^{inc}.

Treatment of leaf nodes in T^{inc}: Let ℓ be a leaf in T^{inc}, X_ℓ^{inc} the vertices of V collected in the box for ℓ and $\hat{X}_\ell^{inc}$ those vertices which only occur in X_ℓ^{inc} but not further up in the boxes of T^{inc}. Recall that according to our general assumptions at least one such vertex exists. For a vertex $i \in V$ let $T^{dec}(i)$ denote the subtree of T^{dec} induced by those nodes that are related to a box containing i. The edges of any cycle cover of G can be split in two groups such that one of them contains only edges that either occur in T_ℓ^{inc} or are incident to an $i \in \hat{X}_\ell^{inc}$ in the graph represented by $T^{dec}(i)$ (i.e., decreasing edges incident with i). We collect this information bottom up to compute all cycle covers and their corresponding contribution to the permanent. Edges between vertices occurring as well further up in T^{inc}, i.e., edges in $(X_\ell^{inc} \setminus \hat{X}_\ell^{inc})^2$, are treated later on. Let C^{inc} denote a partial cycle cover consisting of increasing edges of the graph induced by X_ℓ^{inc}. It assigns to each vertex $i \in X_\ell^{inc}$ some 0-1 information about its in- and out-degree in the partial cover. The number off different such partial covers is bounded as a function in k and all of them can be computed efficiently. For each vertex $i \in \hat{X}_\ell^{inc}$ consider the subtree $T^{dec}(i)$ it induces in T^{dec} and its corresponding graph. Compute from this subtree the set $Dec(i)$ of decreasing edges incident with i. Given the perfectness condition for T^{dec} the number of decreasing edges we obtain is bounded by a function depending on k only. Now again join every C^{inc} with each subset of $Dec(i)$ for all $i \in \hat{X}_\ell^{inc}$ as long as the resulting set of edges gives a partial cycle cover of G.

This way we obtain a list of partial cycle covers of G. To each list there corresponds a 0-1 vector indicating the in- and out-degree of the vertices with respect to a given partial cover. We call such a list together with the additional information a *type* of node ℓ. If λ_ℓ is a type of node ℓ we attach to it as its weight $w(\lambda_\ell)$ the product of all edge-weights for those edges occurring in the corresponding partial cycle cover. The number of types for a leaf is clearly bounded by a function $f(k)$ which once again only depends on the tree-width. The algorithm computes all types for the leaves of T^{inc}. Then it deletes from all boxes in T^{dec} those vertices i that occur in leaf nodes of T^{inc} but not further up. This way i does not any longer influence the algorithm when climbing up, i.e., no further backtracking with respect to such an i is necessary. The new tree we obtain for the remaining decreasing edges is again denoted by T^{dec}.

Climbing up T^{inc}: Next we have to explain how two types are combined when the corresponding nodes have the same predecessor in T^{inc}. Let X_t^{inc} be the box related to an inner node t, let ℓ and r denote the successor nodes of t in T^{inc} and X_ℓ^{inc}, X_r^{inc} the related boxes of vertices from V, respectively. Nodes with one successor only are treated similarly. The goal is to compute types for X_t^{inc} by joining in an appropriate way types for X_ℓ^{inc} and X_r^{inc}. The joining procedure requires several things: First, the types have to represent families of partial covers that fit to the corresponding 0-1 information attached to the type. Secondly, new information given by X_t^{inc} and by T^{dec} for those vertices of V that finally occur in X_ℓ^{inc} or X_r^{inc} has to be incorporated.

Let $\hat{X}_t^{inc}$ denote the set of vertices which when climbing up T^{inc} in box X_t^{inc} occur for the last time. Let λ_t be a type related as before to a partial cycle cover including only edges incident with a vertex in $\hat{X}_t^{inc}$. Note that for increasing edges this says that we do not yet consider edges that occur further up in T^{inc}, and for decreasing edges we incorporate once again the information concerning vertices in $\hat{X}_t^{inc}$ that is represented by T^{dec}. As for the leaves there are at most $f(k)$ many types λ_t obtained that way which are related to X_t^{inc}. Let $E(\lambda_t)$ denote the edges occurring in the partial cycle cover given by λ_t and $w(E(\lambda_t))$ the product of their weights. We next compute in which way such a λ_t fits together with types λ_ℓ and λ_r for the successor boxes X_ℓ^{inc} and X_r^{inc}. We call such a triple compatible if the following conditions hold:

i) when joining the edges of partial cycle covers represented by $\lambda_\ell, \lambda_r, \lambda_t$ we obtain a new partial cycle cover;
ii) all vertices that have been removed already have in- and out-degree 1 in the cover.

Compatibility of three types easily can be checked given the 0-1 information about the in- and out-degree 1 of vertices. Moreover, information concerning vertices that have been removed already does not have to be considered further when climbing up T^{inc}. Since there are no more than $f(k)$ types λ_t for a node t of T^{inc} there are at most $f(k)^2$ many compatible triples. We assign to a type λ_t the weight

$$w(\lambda_t) := w(E(\lambda_t)) \cdot \sum_{\substack{(\lambda_\ell,\lambda_r,\lambda_t)\\ \text{compatible}}} w(\lambda_\ell) \cdot w(\lambda_r) \ .$$

This sum is computable in constantly many steps (depending on k). Finally, we remove the vertices in $\hat{X}_t^{inc}$ from all boxes in T^{dec} and continue climbing up, now using the 0-1 information finally computed for a type λ_t together with its weight $w(\lambda_t)$. Note that both the maximal number of types and the complexity to compute $w(\lambda_t)$ remains the same on each level. Already removed vertices and the information how they occur in particular cycle covers has not to be listed further.

Finishing the computation at the root: When the root s of T^{inc} is reached the algorithm has to check which of the types obtained so far correspond to a family of complete cycle covers of G. This is the case if the vertices still present get in- and out-degree 1 by the type. The latter once again can be checked by inspecting the partial cycle covers containing increasing edges in $X_s^{inc} \times X_s^{inc}$ and the corresponding decreasing edges given by (the remaining vertices of) T^{dec}. The permanent of M then is the sum of the weights of those types representing complete cycle covers.

The algorithm can be performed by an algebraic circuit over $\mathbb{K}$ of logarithmic depth: The inputs for the circuit are the weights m_{ij} of edges e_{ij} given by M. The circuit is constructed of subcircuits which perform the calculations necessary at each node of the tree-decompositions explained above. Since the amount of work for each such node is only depending on k each such subcircuit has constant size (depending as well on k). Given that the tree-decompositions have logarithmic depth the resulting circuit has logarithmic depth as well. Thus, by folklore arguments on circuits the algorithm has a running time that is bounded by a polynomial in n. The above remark concerning the number of types on each level implies that the exponent of this polynomial is independent of k. □

3.2 The General Case

In the above proof the perfectness condition on T^{dec} is important in order to guarantee that a vertex i which has been seen for the last time when climbing up T^{inc} with a restricted amount of additional work can be removed completely as well with respect to edges in G^{dec}. We next show that in relation to the permanent problem this perfectness condition always can be achieved to hold. More precisely, we show the following: Let M be an $n \times n$ matrix, $G = (V, E)$ its weighted adjacency graph and σ a permutation of V for which G_σ^{inc} and G_σ^{dec} are of bounded tree-width. Suppose $T_\sigma^{inc}, T_\sigma^{dec}$ to be the corresponding tree-decompositions. We want to impose the perfectness condition on T_σ^{dec}. Towards this aim we shall construct a new matrix $\tilde{M}$, a corresponding graph $\tilde{G}$ and a permutation $\tilde{\sigma}$ such that $perm(\tilde{M}) = perm(M), \tilde{G}$ is of bounded triangular tree-width witnessed by $\tilde{\sigma}$ and $(\tilde{T}_\sigma^{inc}, \tilde{T}_\sigma^{dec})$ satisfies the perfectness condition via $\tilde{T}_\sigma^{dec}$. Here, the new graph is obtained by adding at most linearly many vertices and edges in such a way that original vertices occurring too often in boxes of T_σ^{dec} are partially replaced by new ones. The replacement has to be done in such a way that the cycle covers of the old and those of the new graph are in a 1-1 correspondence.

Theorem 2. *a) Let $M \in \mathbb{K}^{n \times n}; G, \sigma$ be of triangular tree-width k and this is witnessed by σ. Let $T_\sigma^{inc}, T_\sigma^{dec}$ be related tree-decompositions of $G_\sigma^{inc}, G_\sigma^{dec}$, respectively. Then there are a matrix $\tilde{M}$, a related graph $\tilde{G} = (\tilde{V}, \tilde{E})$ and a permutation $\tilde{\sigma}$ of $\tilde{V}$ such that*

- *$\tilde{G}$ is of triangular tree-width at most $4k + 3$, and this is witnessed by $\tilde{\sigma}$ and decompositions $\tilde{T}_{\tilde{\sigma}}^{inc}, \tilde{T}_{\tilde{\sigma}}^{dec}$;*
- *the pair $(\tilde{T}_{\tilde{\sigma}}^{inc}, \tilde{T}_{\tilde{\sigma}}^{dec})$ is perfect (see Definition 5);*
- *$perm(\tilde{M}) = perm(M)$.*

Moreover, all objects related to $\tilde{M}$ can be computed in polynomial time in n from the given objects attached to G.

b) Let $k \in \mathbb{N}, \{M_i\}_{i \in I}$ a family of matrices of bounded triangular tree-width at most k. For every member M of the family, given corresponding tree-decompositions of the adjacency graphs, $perm(M)$ can be computed in polynomial time in the size of M. Again, the exponent of this polynomial is independent of k.

Proof. Without loss of generality we try to impose the perfectness condition on T_σ^{dec}. For each node i of the original graph G we do the following. If i occurs in at most $10k$ of the boxes related to the nodes of T_σ^{dec}, then i and the corresponding edges remain unchanged. So let i be a node that occurs in more boxes. We iteratively reduce its occurrences according to the following algorithm. This procedure if necessary is repeated several times.

Let t be a node of T_σ^{dec} such that i occurs in at most $10k$ many boxes of the subtree T_t^{dec} rooted at t but in more than $10k$ boxes of the subtree rooted at t's parent. Such a node must exist and can be found easily inspecting T_σ^{dec}. The idea is to introduce two new nodes i_1, i_2 which replace i in all boxes related to nodes below t. In an updated version of X_t^{dec} itself all three nodes i, i_1, i_2 are placed. Replacement of edges is done as follows. We introduce new decreasing edges (i_2, i) and (i, i_1) as well as two loops $(i_1, i_1), (i_2, i_2)$ (recall that loops by convention are considered as decreasing edges). All four edges are given the weight 1. These edges are added to box X_t^{dec}. We extend the order given by the original permutation σ of V to a permutation $\tilde{\sigma}$ of $V \cup \{i_1, i_2\}$ and its induced ordering by placing i_1 immediately before and i_2 immediately after i in the new order. In the boxes below X_t^{dec} we replace each decreasing edge $(j, i), i <_\sigma j$ by (j, i_2) and each decreasing edge $(i, j), i >_\sigma j$ by (i_1, j). To both edges we attach the same weights the original edges had. Finally, i is removed from the boxes occurring below X_t^{dec}. Nothing is changed in T_σ^{inc}; in particular, the new nodes do neither occur in any of its boxes nor are they incident with any increasing edges.

To justify the above construction we explain how partial cycle covers in G that involve vertex i and edges covered in boxes below X_t^{dec} are transformed to partial cycle covers in the new graph. There are several cases to consider.

Suppose first such a partial cover to contain two decreasing edges (j_2, i) and (i, j_1) in boxes below X_t^{dec}. These edges are replaced by the sequence of decreasing edges $(j_2, i_2), (i_2, i), (i, i_1), (i_1, j_1)$. The product of its weights remains the same as that of the weights of $(j_2, i), (i, j_1)$. Next consider edges (j_2, i) and (i, j_1) where the first is decreasing and the second increasing with respect to σ. This edge sequence in a cycle cover is replaced by $(j_2, i_2), (i_2, i), (i, j_1), (i_1, i_1)$, again with the same product of weights. Note here that the increasing edge (i, j_1) involving i does not have to be replaced. It still occurs in T_σ^{inc}. As consequence, i_1 has to be covered by a loop. Thirdly, if the original partial cycle cover involves i in two increasing edges $(j_1, i), (i, j_2), j_1 <_\sigma i <_\sigma j_2$ the two edges are maintained and loops for i_1, i_2 are added. Next, if in the original partial cycle cover we have an increasing edge (j_1, i) and a decreasing one (i, j_2) we replace it by $(j_1, i), (i, i_1), (i_1, j_2), (i_2, i_2)$. Finally, if i is covered by a loop the loop is maintained and one for both i_1 and i_2 is added.

Note that each partial cycle cover in the new graph which fully covers i, i_1, and i_2, i.e., which assigns to all three nodes in- and out-degree 1 origins from exactly one partial cycle cover in the original graph which fully covers i and contributes the same weight. This is correct since there are no increasing edges incident to the newly introduced vertices.

After this part of the construction has been performed we obtain a new graph with two additional vertices occurring in $10k$ boxes of the updated tree $T_{\tilde{\sigma}}^{dec}$. The original vertex i occurs in $10k - 1$ less many boxes as before. The tree T_σ^{inc} remains unchanged and $T_{\tilde{\sigma}}^{dec}$ has the same structure as before, but its tree-width has increased by at most 2 since i, i_1, i_2 commonly occur in the new version of X_t^{dec}.

This process is continued with the new graph and the updated version $T_{\tilde{\sigma}}^{dec}$. First, the number of occurrences of i by the same method is reduced until it occurs in at most $10k$ boxes. For additional new vertices $i_3, i_4, \ldots$ only their order with respect to the original vertices of V matters; the ordering among $i_1, i_3, i_5, \ldots$ and among $i_2, i_4, i_6, \ldots$ is of no concern since there are no edges added between. The previously treated boxes of $T_{\tilde{\sigma}}^{dec}$ below node t remain unchanged; only in X_t^{dec} itself vertex i possibly has to be removed as well during a later phase of the algorithm. This happens at most once. Thus, each vertex i in a box of the original tree T_{σ}^{dec} is replaced by at most 4 new vertices.

Then the other vertices of V are handled accordingly. The algorithm results in a new $m \times m$ matrix $\tilde{M}$, where $m \in O(n)$, a corresponding adjacency graph $\tilde{G}$, a permutation $\tilde{\sigma}$ of the enlarged vertex set $\tilde{V}$ that extends σ and a perfect triangular tree-decomposition $(\tilde{T}_{\tilde{\sigma}}^{inc}, \tilde{T}_{\tilde{\sigma}}^{dec})$ of $\tilde{G}$ with respect to $\tilde{\sigma}$ of triangular tree-width $\leq 4k+3$. Here, $\tilde{T}_{\tilde{\sigma}}^{inc} = T_{\sigma}^{inc}$ and $\tilde{T}_{\tilde{\sigma}}^{dec}$ can be computed from T_{σ}^{dec} in polynomial time in the size of M. Finally, the cycle covers of M and $\tilde{M}$ are in 1-1 correspondence maintaining as well their weights. It follows $perm(\tilde{M}) = perm(M)$.

b) The claim follows from part a) and Theorem 1. Given M and the related triangular tree-decomposition we first apply part a) to obtain $\tilde{M}, \tilde{G}$ and a perfect triangular tree-decomposition. Then, the algorithm behind the proof of Theorem 1 is applied to $\tilde{M}, \tilde{G}$ to compute $perm(\tilde{M})$. Again part a) guarantees the computed value to equal $perm(M)$. The algorithm runs in polynomial time in the size of M since this holds both for the computation of permanents of families of matrices with a bounded perfect triangular tree-decomposition and the algorithm that reduces the general case to the former. □

Remark 1. Note that the earlier mentioned example of triangular matrices does not really completely fit into our framework since such a matrix (if the upper triangular part is fully filled with non-zero entries) as well has unbounded triangular tree-width. Nevertheless the reason why computing the permanent is easy in this case, namely that a cycle cover not existing completely of loops would have to include decreasing edges once it uses an increasing edge, gives the right idea for introducing the triangular tree-width. The example would be formally covered by Theorem 2 if we would speak of bounded triangular tree-width already in case one of the two attached graphs has tree-width 0. As kindly pointed out by one referee one could as well redefine triangular tree-width as $\min_{\sigma \in S_n} twd(G_{\sigma}^{inc}) \cdot twd(G_{\sigma}^{dec})$ which would cover the above case and otherwise increase the ttw parameter at most quadratically.

3.3 Efficiently Solving Subclasses of Hamiltonian Cycle

The idea of distinguishing increasing and decreasing edges applies as well to the Hamiltonian Cycle problem. In this subsection we show that also this problem is efficiently solvable on the subclass of directed graphs with bounded triangular tree-width at most k. Since most of the arguments from before can be used we just briefly point out those that have to be changed. They are related to the task of achieving a perfect triangular decomposition.

Theorem 3. *Let $k \in \mathbb{N}$ be fixed, $\{G_i\}_{i \in I}$ a family of directed graphs with triangular tree-width at most k. Given a member G of the family together with a permutation σ and the corresponding triangular tree-decomposotion of G the Hamiltonian Cycle problem can be solved in polynomial time in the size of G. This holds both for the decision and the counting version.*

Proof. We only describe the necessary changes in the proofs of Theorems 1 and 2. It is appropriate to start with the latter since as result of the necessary adjustments the task of what has to be checked afterwards changes slightly. Let $G, \sigma, G_\sigma^{inc}, G_\sigma^{dec}, T_\sigma^{inc}, T_\sigma^{dec}$ be as before. Again, the perfectness condition is imposed on T_σ^{dec}. However, by doing so we slightly alter the problem to be considered on the new graph $\tilde{G}$ obtained. This is due to the introduction of new vertices which do not in all cases fit within a Hamiltonian cycle of the graph. The new vertices and edges are introduced exactly as before to make T_σ^{dec} perfect. Let V' be the newly introduced vertices. In the new graph $\tilde{G}$ we now do not look for Hamiltonian cycles on $V \cup V'$. Instead we are interested in a potential mixture of a Hamiltonian cycle on a subset of vertices and a cycle cover in the following sense. Call a *partial* Hamiltonian cycle a collection of edges in $\tilde{G}$ such that all vertices of V and may be some of V' are covered by a Hamiltonian cycle on these vertices and the remaining vertices from V' are covered by loops. It is easy to see that perfectness can be gained for T_σ^{dec} in such a way that there is a 1-1 correspondence between Hamiltonian cycles in G and partial Hamiltonian cycles in the new graph $\tilde{G}$. In a second step the algorithm proving Theorem 1 can now be adapted in such a way that when climbing bottom up only types are considered that still potentially might result in a partial Hamiltonian cycle of $\tilde{G}$. □

4 Open Questions

We have introduced an extended version of the tree-width parameter tailored to computing permanents of matrices. The main result of the paper shows that this can be done efficiently for families of matrices where the newly defined parameter is bounded. Though triangular tree-width is defined for directed graphs a main feature is the dependence of the definition on an ordering of the vertices. It thus seems unclear whether there is a relation to the directed tree-width notion introduced in [9]. Another interesting notion to study in this context might be the entaglement width from [3].

There are some questions arising immediately from the issues discussed in this paper. Are there other graph-related problems which in general form are hard to solve but turn out to be easy when the underlying graphs are restricted with respect to their triangular tree-width? For problems which like the above studied ones depend on cycle covers the approach likely will be applicable as well. For other problems this seems much more unclear. For example, consider the problem of counting satisfying assignments of conjunctive normal form formulas. One can naturally assign a directed graph to such a formula by taking as nodes the variables and the clauses and joining a variable and a clause node by an edge if the former occurs in the latter, the direction of an edge depends on whether a variable or its negation occurs in the clause. As it is shown in [7] this problem is solvable efficiently if the latter graph is of bounded tree-width. So the question is whether this still remains to be the case for a familiy of formulas whose graphs have bounded triangular tree-width?

Another related question is whether a similar approach as well could be used for defining extensions of other width notions. Here, clique width would be the first choice to be analyzed.

Acknowledgement. Thanks are due to the anonymous referees for careful reading of the paper and some very helpful comments.

References

1. Arnborg, S., Corneil, D., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM Journal on Matrix Analysis and Applications 8(2), 277–284 (1987)
2. Barvinok, A.: Two algorithmic results for the traveling salesman problem. Mathematics of Operations Research 21, 65–84 (1996)
3. Berwanger, D., Grädel, E.: Entanglement – A Measure for the Complexity of Directed Graphs with Applications to Logic and Games. In: Baader, F., Voronkov, A. (eds.) LPAR 2004. LNCS (LNAI), vol. 3452, pp. 209–223. Springer, Heidelberg (2005)
4. Bodlaender, H.L.: NC-algorithms for graphs with small tree-width. In: Proc. Graph-theoretic concepts in computer science. LNCS, vol. 344, pp. 1–10. Springer, Heidelberg (1989)
5. Briquel, I., Koiran, P., Meer, K.: On the expressive power of CNF formulas of bounded tree- and clique-width. Discrete Applied Mathematics (to appear)
6. Courcelle, B., Makowsky, J.A., Rotics, U.: On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. Discrete Applied Mathematics 108(1-2), 23–52 (2001)
7. Fischer, E., Makowsky, J., Ravve, E.V.: Counting Truth Assignments of Formulas of Bounded Tree-Width or Clique-Width. Discrete Applied Mathematics 156, 511–529 (2008)
8. Flarup, U., Koiran, P., Lyaudet, L.: On the expressive power of planar perfect matching and permanents of bounded treewidth matrices. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 124–136. Springer, Heidelberg (2007)
9. Johnson, T., Robertson, N., Seymour, P.D., Thomas, R.: Directed Tree-Width. Journal Comb. Theory, Series B 82, 138–154 (2001)
10. Thomassen, C.: Handbook of Combinatoris. In: Grötschel, M., Lovász, L., Graham, R.L. (eds.) Embeddings and Minors, pp. 302–349. North-Holland, Amsterdam (1995)
11. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science 8(2), 189–201 (1979)
12. Valiant, L.G.: Completeness classes in algebra. In: Proc. 11th ACM Symposium on Theory of Computing 1979, pp. 249–261. ACM, New York (1979)

Computing Vertex-Surjective Homomorphisms to Partially Reflexive Trees[⋆]

Petr A. Golovach, Daniël Paulusma, and Jian Song

School of Engineering and Computing Sciences, Durham University,
Science Laboratories, South Road, Durham DH1 3LE, UK
{petr.golovach,daniel.paulusma,jian.song}@durham.ac.uk

Abstract. A homomorphism from a graph G to a graph H is a vertex mapping $f : V_G \to V_H$ such that $f(u)$ and $f(v)$ form an edge in H whenever u and v form an edge in G. The H-Coloring problem is to test whether a graph G allows a homomorphism to a given graph H. A well-known result of Hell and Nešetřil determines the computational complexity of this problem for any fixed graph H. We study a natural variant of this problem, namely the Surjective H-Coloring problem, which is to test whether a graph G allows a homomorphism to a graph H that is (vertex-)surjective. We classify the computational complexity of this problem when H is any fixed partially reflexive tree. Thus we identify the first class of target graphs H for which the computational complexity of Surjective H-Coloring can be determined. For the polynomial-time solvable cases, we show a number of parameterized complexity results, especially on nowhere dense graph classes.

1 Introduction

A graph is denoted $G = (V_G, E_G)$, where V_G is the set of vertices and E_G is the set of edges. A *homomorphism* from a graph G to a graph H is a mapping $f : V_G \to V_H$ that maps adjacent vertices of G to adjacent vertices of H, i.e., $f(u)f(v) \in E_H$ whenever $uv \in E_G$.

The problem H-Coloring tests whether a given graph G allows a homomorphism to a graph H called the *target*. Throughout our paper we assume that H denotes a *fixed* graph (i.e., not part of the input) except when we consider a parameterized setting and choose $|V(H)|$ as the parameter. If H is the complete graph (graph with edges between all pairs of different vertices) on k vertices, then this problem is equivalent to the k-Coloring problem, which is to test whether a graph G allows a mapping $c : V_G \to \{1, \ldots, k\}$ such that $c(u) \neq c(v)$ whenever $uv \in E_G$.

For a survey on homomorphisms we refer to Hell and Nešetřil [15]. Here, we only mention the classical result in this area, which is the Hell-Nešetřil dichotomy theorem [14]. This theorem states that H-Coloring is solvable in polynomial time if H is bipartite, and NP-complete otherwise. Note that H is assumed to have no self-loop xx, as otherwise we can map every vertex of G to x.

[⋆] This work has been supported by EPSRC (EP/G043434/1).

A homomorphism f from a graph G to a graph H is *surjective* if for each $x \in V_H$ there exists at least one vertex $u \in V_G$ with $f(u) = x$. This paper studies the problem of deciding if a given graph allows a surjective homomorphism to a target graph H. This problem is called the SURJECTIVE H-COLORING problem. We observe that, for this variant, the presence of a vertex with a self-loop in the target graph H does not make the problem trivial. So, we do allow such vertices in H and call them *reflexive*, whereas vertices with no self-loop are said to be *irreflexive*. A graph that contains zero or more reflexive vertices is called *partially reflexive*. In particular, a graph is *reflexive* if all its vertices are reflexive, and a graph is *irreflexive* if all its vertices are irreflexive. Throughout the paper, we assume that the input graph G is irreflexive and that the target graph H is partially reflexive. We also assume that both graphs are undirected, finite and have no multiple edges.

In contrast to the SURJECTIVE H-COLORING problem, the injective variant has been well studied in the literature; when both G and H are part of the input, the injective variant is equivalent to the SUBGRAPH ISOMORPHISM problem. Below we discuss a number of other problems that are closely related to SURJECTIVE H-COLORING.

Locally surjective homomorphisms. A homomorphism f from a graph G to a graph H is *locally surjective* if f becomes surjective when restricted to the neighborhood of every vertex u of G. We also say that such an f is an *H-role assignment*, and the corresponding decision is called the H-ROLE ASSIGNMENT problem. Any locally surjective homomorphism is surjective if the target graph is connected but the reverse implication is not true in general.

The computational complexity of the H-ROLE ASSIGNMENT problem has been completely classified with the problem being solvable in polynomial time if and only if the fixed graph H has no edge, or H has an isolated reflexive vertex, or H is bipartite, irreflexive and with an isolated edge. In all other cases, H-ROLE ASSIGNMENT is NP-complete [13]. For more on locally surjective homomorphisms and the locally injective and bijective variants, we refer to the survey of Fiala and Kratochvíl [12].

List-homomorphisms and retractions. Let G and H be two graphs with a list $L(u) \subseteq V_H$ associated to each vertex $u \in V_G$. Then a homomorphism f from G to H is a *list-homomorphism* with respect to the lists L if $f(u) \in L(u)$ for all $u \in V_G$. List-homomorphisms were introduced by Feder and Hell [8] and generalize list-colorings. Feder, Hell and Huang [9] completely classified the computational complexity of the problem that tests whether a graph G with lists L allows a list-homomorphism to a fixed graph H with respect to lists L. In our context, a special kind of list homomorphisms are of importance, namely the retractions defined below.

Let H be an induced subgraph of a graph G. A homomorphism f from a graph G to H is a *retraction* from G to H if $f(h) = h$ for all $h \in V_H$. In that case we say that G *retracts to* H. A retraction from G to H can be viewed as a list-homomorphism if we choose $L(x) = \{x\}$ for each $x \in V_H$ and $L(u) = V_H$ for each $u \in V_G \setminus V_H$.

The H-Retraction problem asks if a graph G retracts to a fixed subgraph H. A *pseudoforest* is a graph in which each component has at most one cycle different from a self-loop. Feder et al. [10] classified the H-Retraction problem for all fixed pseudoforests H.

Compactions. We stress that a surjective homomorphism is *vertex-surjective* as opposed to the stronger condition of being *edge-surjective*. The latter condition has been defined in the literature as well. A homomorphism from a graph G to a graph H is called *edge-surjective* or a *compaction* if for any edge $xy \in E_H$ with $x \neq y$ there exists an edge $uv \in E_G$ with $f(u) = x$ and $f(v) = y$. Note that the edge-surjectivity condition only holds for edges $xy \in E_H$; there is no such condition on the self-loops $xx \in E_H$. If f is a compaction from G to H, we also say that G *compacts* to H.

The H-Compaction problem asks if a graph G compacts to a fixed graph H. Vikas [21–23] determined the computational complexity of this problem for several classes of fixed target graphs, e.g., when H is a reflexive cycle, an irreflexive cycle, or a graph on at most 4 vertices.

Our Results. We give a complete classification of the computational complexity of the Surjective H-Coloring problem when H is a tree. Because we consider target graphs that are partially reflexive, we stress that H is as a matter of fact a *partially reflexive tree*, i.e., a connected graph with no cycles different from a self-loop. Let R_H denote the (possibly empty) set of reflexive vertices of a graph H. We say that H is *loop-connected* if R_H induces a connected subgraph of H. Our main result is the following theorem:

Theorem 1. *For any fixed tree H, the* Surjective H-Coloring *problem is polynomial time solvable if H is loop-connected, and* NP*-complete otherwise.*

We analyze the running time of the polynomial-time solvable cases in Theorem 1. For connected n-vertex graphs we find a running time of $n^{k+O(1)}$, where k is the number of leaves of H. We show that there is no function f that only depends on k such that this running time can be improved to $f(k) \cdot n^{O(1)}$, unless FPT = W[1], or to $f(k) \cdot n^{o(k)}$, unless the Exponential Time Hypothesis [16] collapses. On the positive side, we prove that for any loop-connected tree H, Surjective H-Coloring parameterized by $|V_H|$ is FPT on any nowhere dense graph class. Examples of such graph classes are graphs of bounded genus (e.g. planar graphs), graphs that exclude a fixed (topological) minor and graphs that locally exclude a fixed minor [18].

2 The Polynomially Solvable Cases of Theorem 1

We use the classification of Feder et al. [10] on the H-Retraction problem when H is a pseudoforest.

Theorem 2 ([10]). *For a fixed pseudoforest H, the H-*Retraction *problem is* NP*-complete if (i) H contains a component that is not loop-connected, or* *(ii) H contains a cycle on at least 5 vertices, or (iii) H contains a reflexive cycle on 4 vertices, or (iv) H contains an irreflexive cycle on 3 vertices. In all other cases, the H-*Retraction *problem can be solved in polynomial time.*

Let G be a graph. Let $U \subseteq V_G$. We let $G[U]$ denote the subgraph of G induced by U. We also need the following result.

Proposition 1. *Let H be a fixed partially reflexive graph. If the H-*Retraction *problem can be solved in $f(n, |V_H|)$ time on n-vertex graphs, then the* Surjective H-Coloring *problem can be solved in time $n^{|V_H|+O(1)} \cdot f(n, |V_H|)$.*

Proof. Let $V_H = \{x_1, \ldots, x_\ell\}$. Let G be a graph on n vertices. We guess an ordered set $U = \{u_1, \ldots, u_\ell\}$ of vertices of G and map u_i to x_i for $i = 1, \ldots, \ell$. We check if $x_i x_j \in E_H$ whenever $u_i u_j \in E_G$. If not we guess a different set U. Otherwise, we add an edge $u_i u_j$ whenever $x_i x_j \in E_H$ and $u_i u_j \notin E_G$. This leads to a graph G' such that $G'[U]$ is isomorphic to H. We solve H-Retraction on G'. If we find a retraction f, then f is a surjective homomorphism from G to H. Otherwise we guess a different ordered set U. Because there are at most $n^{|V_H|}$ different sets U and constructing G' costs $n^{O(1)}$ time, the result follows. □

Combining Theorem 2 and Proposition 1 proves the polynomial part of Theorem 1; in fact we have a bit stronger result.

Corollary 1. *For a pseudoforest H,* Surjective H-Coloring *can be solved in polynomial time if every component of H is loop-connected, and H contains no cycle on at least 5 vertices, no reflexive cycle on 4 vertices, and no irreflexive cycle on 3 vertices.*

2.1 Parameterized Complexity

Note that Corollary 1 does not give any exact running times; Feder et al. [10] do not state the exact running time of their polynomial-time algorithm in Theorem 2. As a side effect of the proof of our FPT result on nowhere dense graph classes we obtain the following, a proof of which will be given later (see Remark 1). Let H be a loop-connected tree with k leaves. Then Surjective H-Coloring can be solved in $n^{k+O(1)}$ time on n-vertex connected graphs. Our next result shows that there is no function f that only depends on k such that this running time can be improved to $f(k) \cdot n^{O(1)}$, unless FPT = W[1]; see e.g. Downey and Fellows [6] for definitions of these parameterized complexity classes.

Let S_k denote the graph obtained from the star $K_{1,k}$ after adding a self-loop to its center. Note that Surjective S_k-Coloring can be solved in polynomial time by Theorem 1 (or Corollary 1). We observe that a connected graph G on at least two vertices allows a surjective homomorphism to S_k if and only if G has an independent set of size at least k. Because the Independent Set problem, which asks whether a graph has an independent set of size at least k, is W[1]-complete when parameterized by k (cf. [6]), we immediately obtain the following.

Proposition 2. Surjective S_k-Coloring *is* W*[1]-complete when parameterized by k.*

The assumption that there is no algorithm that solves the 3-Satisfiability problem in $2^{o(n)}$ time on n-variable formulas is known as the Exponential Time Hypothesis [16]. Our next result shows that the $n^{k+O(1)}$ running time of Surjective H-Coloring for a loop-connected tree H cannot be improved to $f(k) \cdot n^{o(k)}$, unless the Exponential Time Hypothesis collapses. Chen et al. [1] showed that there is no algorithm that solves Independent Set on n-vertex graphs in time $f(k) \cdot n^{o(k)}$, unless the Exponential Time Hypothesis collapses. This gives us the following.

Proposition 3. Surjective S_k-Coloring *cannot be solved in $f(k) \cdot n^{o(k)}$ time on n-vertex graphs, unless the Exponential Time Hypothesis collapses.*

Due to Proposition 2 and 3 it makes sense to consider special graph classes in order to improve the running time. For this purpose we consider graph classes that are nowhere dense, a notion introduced by Nešetřil and Ossona de Mendez [18]. In order to define these graph classes we first need to state some extra terminology.

Let G be a graph. We say that two subgraphs of G are *adjacent* if G has an edge between a vertex from one subgraph and a vertex of the other subgraph. A graph F is a minor of G if F can be obtained from G by a series of edge contractions, edge deletions and vertex deletions. We use an equivalent characterization (see e.g. [5]), namely that F is a minor of G if and only if we can associate each vertex x of F with a tree G_x under the following three conditions: (i) for all $x \in V_F$, the tree G_x is a subgraph of G; (ii) for all $x, y \in V_F$ with $x \neq y$, the trees G_x and G_y are vertex-disjoint; (iii) for all $x, y \in V_F$ with $x \neq y$, the trees G_x and G_y are adjacent if and only if x and y are adjacent in F. The trees G_x are called the *branch sets* for H. A graph F is a *minor at depth* $r \geq 0$ of G if there exists branch sets $\{G_x \mid x \in V_F\}$ for F such that every G_x is a graph of radius at most r. A graph class $\mathcal{G}$ is *nowhere dense* if for every r there is a graph F such that F is not a minor at depth r of G for all $G \in \mathcal{G}$.

Dawar and Kreutzer [4] and Dvorak et al. [7] independently showed that the problem of deciding any property that can be expressed in the first-order logic is FPT on a nowhere dense graph class when parameterized by the length of the sentence defining the property. Due to this, we get our desired result if we can show that the existence of a surjective homomorphism from a graph G to a loop-connected tree H can be reduced to a problem that can be expressed in the first-order logic. This is our objective for the rest of this section.

We need the following terminology. Let G be a graph. We denote the *neighborhood* of a vertex u in G by $N_G(u) = \{v \mid uv \in E_G\}$. For a subset $U \subseteq V_G$, we define $N_G(U) = \{v \mid v \in N_G(u) \setminus U$ for some $u \in U\}$. We say that we *glue* a set $W \subseteq V_G$ into a new vertex w^* if we remove all vertices of W and add w^* to G by making it adjacent to every vertex in $N_G(W)$. The following observation follows immediately from the definition of a surjective homomorphism.

Observation 1. *Let G and H be two graphs and let $h : V_G \to V_H$ be a mapping. Let $x \in V_H$ and let $W \subseteq h^{-1}(x)$. Let G' be the graph obtained from G by gluing W into w^*. Let $h'\colon V_{G'} \to V_H$ be the mapping defined as*

$$h'(v) = \begin{cases} h(v), & v \neq w^*, \\ x, & v = w^*. \end{cases}$$

Then the following holds:

- *if h is a surjective homomorphism from G to H then h' is a surjective homomorphism from G' to H;*
- *if h' is a surjective homomorphism from G' to H, and W is independent or else x is reflexive then h is a surjective homomorphism from G to H.*

Let v be a vertex of a partially reflexive tree H rooted at r. Observe that r defines the parent-child relation between any two adjacent vertices. Then $C(v)$ denotes the set of all children of v, and $D(v) \supseteq C(v)$ denotes the set of all descendants of v. We assume that both sets are proper, i.e., $v \notin D(v)$, and consequently $v \notin C(v)$.

Definition 1. *Let H be a tree that has a reflexive root r. Let $L = \{z_1, \dots, z_k\}$ consist of all leaves of H that are not equal to r. Let $U = \{u_1, \dots, u_k\}$ be a subset of vertices of a graph G. The mapping $f_U\colon V_G \to V_H$ is given by $f_U(v) = x$ if $v \in W_x$, where W_x is a subset of V_G defined inductively:*

1. *Set $W_{z_i} = \{u_i\}$ for $i = 1, \dots, k$.*
2. *Let x be in $V_H \setminus (\{r\} \cup L)$. Assume that sets W_y are constructed for all $y \in D(x)$. Let $Z \subseteq V_H$ be the set of all vertices of H such that sets W_z are constructed for $z \in Z$. We set $W_x = \bigcup_{y \in C(x)} N_G(W_y) \setminus \bigcup_{z \in Z} W_z$.*
3. *Finally, to define W_r, we assume that sets W_x are constructed for all $x \in V_H \setminus \{r\}$, and set $W_r = V_G \setminus \bigcup_{x \in D(r)} W_x$.*

The following lemma is the first of two crucial lemmas.

Lemma 1. *Let H be a loop-connected tree with reflexive root r and with set of leaves $L = \{z_1, \dots, z_k\}$ not equal to r. Let h be a surjective homomorphism from a connected graph G to H. Let $U = \{u_1, \dots, u_k\} \subseteq G$ be such that $h(u_i) = z_i$ for $i = 1, \dots, k$. Then f_U is a surjective homomorphism from G to H.*

Proof. We use induction on V_H. The statement of the lemma is true if $|V_H| = 1$. Suppose $|V_H| \geq 2$.

Let $x \in V_H \setminus \{r\}$, such that $\emptyset \neq C(x) \subseteq L$. If such a vertex x does not exist then let $x = r$; observe that $C(r) = L$ because in this case H is a star with a reflexive central vertex. We assume without loss of generality that $C(x) = \{z_1, \dots, z_s\}$ for some $1 \leq s \leq k$.

We may without loss of generality assume that $h^{-1}(z_i) = \{u_i\}$ for $i = 1, \dots, s$. In order to see this, suppose that $h^{-1}(z_i)$ contains at least two vertices for some

$1 \leq i \leq s$. Then h can be redefined as follows. If x is a reflexive vertex then all vertices of $h^{-1}(z_i) \setminus \{u_i\}$ may be mapped to x. Otherwise, if x is irreflexive, then x has a parent y. Because r is reflexive and x is irreflexive, z_i cannot be reflexive; otherwise $H[R_H]$ is disconnected, which is not what we assume. Hence, we may map the vertices of $h^{-1}(z_i) \setminus \{u_i\}$ to y.

If $x = r$ then $h^{-1}(z_i) = \{u_i\}$ for $i = 1, \ldots, s$ implies that $f_U = h$, so the statement of the lemma is true. Suppose that $x \neq r$. Let $W = \bigcup_{i=1}^{s} N_G(u_i)$. Note that $W \neq \emptyset$, because G is connected. We find that every neighbor of every u_i is mapped to x, because x is the only neighbor of z_i and h only maps z_i to u_i, as we deduced above. This means that $h(W) = \{x\}$.

Let G' be the connected graph obtained from G by gluing W into w^*. Then, by Observation 1, the mapping $h' \colon V_{G'} \to V_H$ such that

$$h'(v) = \begin{cases} h(v), & v \neq w^*, \\ x, & v = w^* \end{cases}$$

is a surjective homomorphism from G' to H. Let $G'' = G' - \{u_1, \ldots, u_s\}$, and let $H' = H - \{z_1, \ldots, z_s\}$. Observe that $h'' = h'|_{V_{G''}}$ is a surjective homomorphism from G'' to H'. We also observe that G'' is connected. Let $U' = \{w^*, u_{s+1}, \ldots, u_k\}$. Then, by the induction hypothesis, we find that $f'_{U'}$ is a surjective homomorphism from G'' to H'. By the definition of f_U we find that

$$f_U(v) = \begin{cases} f'_{U'}(v), & v \notin \{u_1, \ldots, u_s\} \cup W, \\ f'_{U'}(w^*), & v \in W, \\ z_i, & v \in \{u_1, \ldots, u_s\}. \end{cases}$$

Suppose that x is reflexive. By Observation 1, we obtain that f_U is a surjective homomorphism from G to H. Suppose that x is irreflexive. As we already deduced, in that case z_i must be irreflexive for $i = 1, \ldots, s$. This means that h maps every vertex of W to x. Consequently, W is an independent set. Again we use Observation 1 to obtain that f_U is a surjective homomorphism from G to H. This completes the proof of Lemma 1. □

Let r' be a neighbor of a root r of a tree H. We say that H is *rooted* by the ordered pair (r, r').

Definition 2. *Let H be an irreflexive tree rooted by (r, r'). Let $L = \{z_1, \ldots, z_k\}$ consist of all leaves of H that are neither equal to r nor to r'. Let $U = \{u_1, \ldots, u_k\}$ be a subset of vertices of a bipartite graph G on partition classes V_1 and V_2. Let $(p, q) \in \{(1, 2), (2, 1)\}$. The mapping $f_U^{p,q} \colon V_G \to V_H$ is given by $f_U^{p,q}(v) = x$ if $v \in W_x$, where W_x is a subset of V_G defined inductively:*

1. *Set $W_{z_i} = \{u_i\}$ for $i = 1, \ldots, k$.*
2. *Let x be a vertex of $V_H \setminus (L \cup \{r, r'\})$. Assume that sets W_y are constructed for all $y \in D(x)$. Let $Z \subseteq V_H$ be the set of all vertices of H such that sets W_z are constructed for $z \in Z$. We set $W_x = \bigcup_{y \in C(x)} N_G(W_y) \setminus \bigcup_{z \in Z} W_z$.*

3. *Finally, to define W_r and $W_{r'}$, we assume that sets W_x are constructed for all $x \in V_H \setminus \{r, r'\}$. We set $W_r = V_p \setminus \bigcup_{z \in Z} W_z$ and $W_{r'} = V_q \setminus \bigcup_{z \in Z} W_z$.*

The next lemma is the second crucial lemma.

Lemma 2. *Let G be a connected bipartite graph with partition classes V_1 and V_2. Let H be an irreflexive tree that is rooted by (r, r'), and let $L = \{z_1, \ldots, z_k\}$ be the set of leaves of H that are neither equal to r nor to r'. Let h be a surjective homomorphism from G to H. Let $U = \{u_1, \ldots, u_k\} \subseteq V_G$ such that $h(u_i) = z_i$ for $i = 1, \ldots, k$. If $h^{-1}(r) \subseteq V_p$ and $h^{-1}(r') \subseteq V_q$ then $f_U^{p,q}$ is a surjective homomorphism from G to H.*

Proof. We use induction on V_H. If $|V_H| \leq 2$ then the statement of the lemma is true. Suppose that $|V_H| \geq 3$.

Let $x \in V_H$ such that $\emptyset \neq C(x) \setminus \{r'\} \subseteq L$. We assume without loss of generality that $C(x) \setminus \{r'\} = \{z_1, \ldots, z_s\}$ for some $1 \leq s \leq k$.

We may without loss of generality assume that $h^{-1}(z_i) = \{u_i\}$ for $i = 1, \ldots, s$. In order to see this, suppose $h^{-1}(z_i)$ contains at least two vertices for some $1 \leq i \leq s$. We redefine h as follows by mapping all vertices of $h^{-1}(z_i) \setminus \{u_i\}$ to y, where y is the parent of x unless $x \in \{r, r\}$; if $x = r$ then we take $y = r'$ and if $x = r'$ then we take $y = r$. The resulting mapping is a surjective homomorphism.

Let $W = \bigcup_{i=1}^{s} N_G(u_i)$. Note that $W \neq \emptyset$, because G is connected. We also observe that $h(W) = x$, because z_i is irreflexive and has x as its only neighbor for $i = 1, \ldots, s$. Let G' be the connected graph obtained from G by gluing W into w^*. Then, by Observation 1, the mapping $h' \colon V_{G'} \to V_T$ defined as

$$h'(v) = \begin{cases} h(v), & v \neq w^*, \\ x, & v = w^* \end{cases}$$

is a surjective homomorphism from G' to H. Let $G'' = G' - \{u_1, \ldots, u_s\}$, and let $H' = H - \{z_1, \ldots, z_s\}$. Observe that $h'' = h'|_{V_{G''}}$ is a surjective homomorphism from G'' to H'. We also observe that G'' is connected. Suppose $h^{-1}(r) \subseteq V_p$ and $h^{-1}(r') \subseteq V_q$. Let $U' = \{w^*, u_{s+1}, \ldots, u_k\}$. Then, by the induction hypothesis, $(f_{U'}^{p,q})'$ is a surjective homomorphism from G'' to H'. By the definition of $f_U^{p,q}$, we find that

$$f_U^{p,q}(v) = \begin{cases} (f_{U'}^{p,q})'(v), & v \notin \{u_1, \ldots, u_s\} \cup W, \\ (f_{U'}^{p,q})'(w^*), & v \in W, \\ z_i, & v \in \{u_1, \ldots, u_s\}. \end{cases}$$

Because $h(W) = x$ and x is irreflexive, W is independent. We use Observation 1 and find that $f_U^{p,q}$ is a surjective homomorphism from G to H. □

We are now ready to prove the main result of this section.

Theorem 3. *Let H be a loop-connected tree. Then* Surjective H-Coloring *is* FPT *for any nowhere dense graph class when parameterized by $|V_H|$.*

Proof. By the result of Dawar and Kreutzer [4] we have proven Theorem 3 after showing that the existence of a surjective homomorphism from G to H can be reduced in FPT-time to a problem that can be expressed in the first-order logic.

Let H be a loop-connected tree. Let G be a graph with components $G_1, \ldots, G_p$ for some $p \geq 1$. Then G allows a surjective homomorphism to H if and only if every G_i allows a surjective homomorphism to some H_i for connected induced subgraphs $H_1, \ldots, H_p$ of H such that $V_H = \bigcup_{i=1}^{p} V_{H_i}$. Because we can construct all possible tuples $(H_1, \ldots, H_p)$ in FPT-time by brute force, we may assume that $p = 1$, i.e., that G is connected.

Recall that the syntax of the first-order logic of graphs includes logical connectives $\vee$, $\wedge$, $\neg$, $\Leftrightarrow$, $\Rightarrow$, variables for vertices, and quantifiers $\forall$, $\exists$ that can be applied to these variables. The syntax also includes the following two binary relations for two vertex variables u and v, namely "adj(u, v)", which expresses whether u and v are adjacent, and "$u = v$", which expresses whether u and v are equal.

We distinguish between the cases $R_H \neq \emptyset$ and $R_H = \emptyset$. First suppose that $R_H \neq \emptyset$. We choose a root vertex r in H, which defines the parent-child relation between every pair of adjacent vertices. We let $\{z_1, \ldots, z_k\}$ be the set of all non-root leaves of H. By Lemma 1, there is a surjective homomorphism of G to H if and only if there is an ordered subset $U = \{u_1, \ldots, u_k\}$ of vertices of G such that f_U is a surjective homomorphism from G to H. By the definition of f_U we find that for every $v \in V_G$ and every $x \in V_H$, the inclusion $v \in W_x$ and therefore the property $f_U(v) = x$ can be expressed in the first-order logic. Now we can express the property that there is an ordered set of vertices $U = \{u_1, \ldots, u_k\}$ of G such that f_U is a surjective homomorphism of G to H:

$\exists u_1 \ldots \exists u_k$ such that $u_i \neq u_j$ if $i \neq j$, and $\forall x \in V_H$ $\exists v \in V_G$ such that $x = f_U(v)$, and $\forall v, w \in V_G$, $v \neq w$, $\exists x, y \in V_H$ such that

- $f_U(v) = x$ and $f_U(w) = y$,
- if $x = y$ then adj(v, w) if and only if $x \in R_H$,
- if $x \neq y$ then adj(v, w) if and only if x, y are adjacent in H.

Now suppose $R_H = \emptyset$. We answer `No` if G is not bipartite, because only bipartite graphs allow a homomorphism to a bipartite graph. Hence, assume that G is bipartite with partition classes V_1 and V_2. Because G is connected, we find that for every homomorphism h from G to H either $h^{-1}(x) \subseteq V_1$ or $h^{-1}(x) \subseteq V_2$ for each $x \in V_H$. Hence, we can use Lemma 2 instead of Lemma 1, and the rest of the analysis is similar to the case when $R_H \neq \emptyset$. □

Remark 1. Lemma 1 and 2 immediately yield an $n^{O(1)}$ time algorithm that solves H-Retraction on a connected n-vertex graph G when H is a loop-connected tree. This can be seen as follows. Let H' denote the induced subgraph of G that is isomorphic to H. Then H' fixes the set U. Suppose $R_H \neq \emptyset$. We observe that the construction of f_U respects H'. Hence, by Lemma 1, we only have to construct f_U and check if the obtained mapping is a surjective homomorphism from G to H. This takes $n^{O(1)}$ time. If $R_H = \emptyset$, we first check whether G

is bipartite, say with partition classes V_1 and V_2, as otherwise the answer is `No`. We also recall that for every homomorphism h from G to H either $h^{-1}(x) \subseteq V_1$ or $h^{-1}(x) \subseteq V_2$ for each $x \in V_H$. Hence we can use Lemma 2 to derive the same running time. Note that the same running time for H-Retraction is obtained if G is not connected. The reason is that H will be an induced subgraph of a component of G, because H is connected. We also observe that this running time can be obtained by analyzing the algorithm of Feder et al. [10]. However, they do not define the mappings f_U and $f_U^{p,q}$ explicitly. We had to do this in order to prove Theorem 3. By Proposition 1, we obtain an $n^{|V_H|+O(1)}$ time algorithm that solves Surjective H-Coloring on an n-vertex graph G when H is a loop-connected tree. If G is connected, we may obtain a considerable improvement. In that case, we consecutively check all ordered k-vertex sets U and apply Lemma 1 or 2, respectively. Because the number of different sets U is $O(n^k)$, we find a total running time of $n^{k+O(1)}$. Note that in the case that $R_H = \emptyset$, we must also consider the pairs $(p,q) = (1,2)$ and $(p,q) = (2,1)$. However, this just yields an extra factor n^2.

3 The NP-Complete Cases of Theorem 1

In this section we show that the Surjective H-Coloring problem is NP-complete for any tree H that is not loop-connected. Because checking if a given mapping is a surjective homomorphism can be done in polynomial time, the problem belongs to NP. In order to prove NP-hardness we will reduce from a variant of the Matching-Cut problem. This problem is to test whether a graph G has a *matching-cut* M, i.e., a matching $M \subseteq E_G$ such that $G - M$ is disconnected. Patrignani and Pizzonia [19] prove that this problem is NP-complete. The variant we need is the following problem:

Matching-Cut with Roots
Instance: A connected graph G of minimum degree at least two, and two vertices s, t of G such that s and t are in two different components of $G - M$ for any matching-cut M (if G has a matching-cut).
Question: Does G have a matching-cut M?

Matching-Cut with Roots is NP-complete. This follows immediately from the reduction of the Not All Equal 3-Satisfiability problem to the Matching Cut problem, as described by Patrignani and Pizzonia [19].

We start with some auxiliary constructions. Let H be a tree that is not loop-connected. The *distance* $\text{dist}_H(x,y)$ between a pair of vertices x and y of H is the number of edges on a shortest path between them. We choose two vertices $p, q \in V_H$ that belong to two different components of $H[R_H]$ in such a way that $\text{dist}_H(p,q) \leq \text{dist}_H(x,y)$ for any pair x, y that are in two different components of $H[R_H]$. Let $\ell = \text{dist}_H(p,q)$. By definition, $\ell \geq 2$. Let H_1 and H_2 be two different components of the forest obtained from H after removing the edge incident with q in the unique p,q-path in H. Assume that $p \in V_{H_1}$ and $q \in V_{H_2}$. We construct graphs F_i for $i = 1, 2$ (see Fig. 1) as follows:

1. For each vertex $x \in V_{H_i} \setminus R_H$, we introduce a vertex $t_x^{(1)}$;
2. For each vertex $x \in V_{H_i} \cap R_H$, we introduce two adjacent vertices $t_x^{(1)}, t_x^{(2)}$;
3. For each edge $xy \in E_{H_i}$, we add an edge between any $t_x^{(h)}$ and any $t_y^{(j)}$.

We say that $t_p^{(1)}, t_p^{(2)}$ are the *roots* of F_1, and $t_q^{(1)}, t_q^{(2)}$ are the *roots* of F_2.

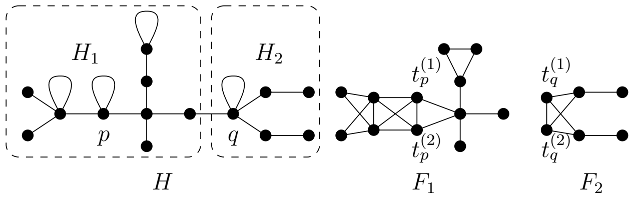


Fig. 1. Construction of F_1 and F_2

Let $\deg_G(u) = |N_G(u)|$ denote the *degree* of a vertex u in a graph G. We now describe our polynomial-time reduction from Matching-Cut with Roots to Surjective H-Coloring. Let G be a connected graph that has minimum degree at least two. Let s, t be two vertices of G that are separated by any matching-cut in G (if a matching-cut exists). From F_1, F_2, and G we construct a graph G' (see Fig. 2) as follows:

1. For each $u \in V_G$ we construct a clique C_u on $\max\{\deg_G(u), 3\}$ vertices if $u \notin \{s,t\}$ and on $\deg_G(u) + 2$ vertices if $u \in \{s,t\}$. We denote $d = \deg_G(u)$ vertices of C_u by $g_{u,e_1}, \ldots, g_{u,e_d}$ to indicate that they correspond to the edges $e_1, \ldots, e_d$ that are incident with u in G. We denote the other vertices in C_u by $g_u^{(1)}$ and $g_u^{(2)}$ if they exist.
2. For each edge $e = uv \in E_G$, the vertices $g_{u,e}, g_{v,e}$ are identified if $\ell = 2$, and the vertices $g_{u,e}, g_{v,e}$ are joined by a path P_e of length $\ell - 2$ if $\ell > 2$. For $\ell = 2$, we let P_e be the single vertex $g_{u,e} = g_{v,e}$.
3. We add F_1 by identifying the roots $t_p^{(1)}, t_p^{(2)}$ with vertices $g_s^{(1)}, g_s^{(2)}$.
4. We add F_2 by identifying the roots $t_q^{(1)}, t_q^{(2)}$ with vertices $g_t^{(1)}, g_t^{(2)}$.

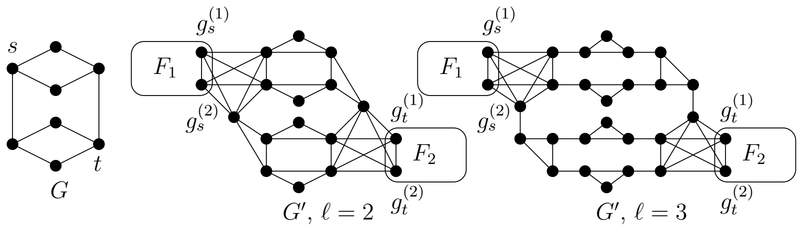


Fig. 2. The construction of G'

Lemma 3. *If G has a matching-cut then there is a surjective homomorphism from G' to H.*

Proof. Let M be a matching-cut in G. Let V_1 be the vertex set of the component of $G - M$ that contains s, and let $V_2 = V_G \setminus V_1$. Note that $t \in V_2$. We define a surjective homomorphism $h : V_{G'} \to V_H$ as follows.

We let h map every vertex in V_1 or V_2 that is not on any path P_e to p or q, respectively. Now consider an edge $e = uv \in E_G$. If u and v are both in V_1 or both in V_2, then we let h map every vertex from P_e except $g_{u,e}$ and $g_{v,e}$ to p or q, respectively. Suppose one of u, v, say u, belongs to V_1, whereas the other one, v, belongs to V_2. Let $P_e = a_1 \cdots a_{\ell-1}$ (notice that $a_1 = g_{u,e}$ and $a_{\ell-1} = g_{v,e}$). Let $px_1 \cdots x_{\ell-1}q$ denote the p, q-path in H. We let h map a_i to x_i for $i = 1, \ldots, \ell-1$. Finally, we let h map every vertex $t_x^{(i)} \in V_{F_1} \cup V_{F_2}$ to x. □

To complete our proof of the NP-complete cases in Theorem 1 we show the following lemma, a proof of which is omitted here due to space restrictions and will appear in the journal paper.

Lemma 4. *If there is a surjective homomorphism of G' to H then G has a matching-cut.*

4 Future Research

We have shown that for any partially reflexive tree H, the Surjective H-Coloring problem is polynomial-time solvable if H is loop-connected and NP-complete otherwise. Determining a complete complexity classification of the Surjective H-Coloring seems a very challenging open problem, and even conjecturing a possible dichotomy (between P and NP-complete) is difficult.

A natural question that also gives an indication on why this problem is so challenging is whether the three problems H-Compaction, H-Retraction and Surjective H-Coloring are polynomially equivalent to each other for each target graph H. Also, the computational complexity classifications of the H-Compaction problem and H-Retraction problem, respectively, are still far from being completed. The well-known Feder-Vardi conjecture [11] states that the $\mathcal{H}$-Constraint Satisfaction problem, where $\mathcal{H}$ is some fixed finite target structure, has a dichotomy. Feder and Vardi [11] showed that this conjecture is equivalent to the conjecture that H-Retraction has a dichotomy.

We will try to extend Theorem 1 in the direction of Theorem 2, i.e., from partially reflexive trees to pseudoforests. An extension to forests is straightforward. However, if we consider target graphs that are cycles then the smallest case that we cannot solve is the case in which H is the reflexive 4-cycle $\mathcal{C}_4$. Although both $\mathcal{C}_4$-Retraction and $\mathcal{C}_4$-Compaction are NP-complete, as proven by Feder et al. [10] and Vikas [21], respectively, the problem Surjective $\mathcal{C}_4$-Coloring is known to be wide open. We will explain this below.

Fleischner et al. [17] showed that Surjective $\mathcal{C}_4$-Coloring is equivalent to the problem Disconnected Cut that is to test whether a graph $G = (V, E)$ has a vertex cut $U \subseteq V$ such that $G[U]$ is disconnected. In particular, they show that every graph of diameter at least three allows a surjective homomorphism to $\mathcal{C}_4$, and that the two problems Surjective $\mathcal{C}_4$-Coloring and $\mathcal{C}_4$-Compaction are equivalent on input graphs of diameter two. However, the reduction of Vikas [21] for $\mathcal{C}_4$-Compaction cannot be used for graphs of diameter two.

The Surjective $\mathcal{C}_4$-Coloring problem is also studied in the context of H-partitions introduced by Dantas et al. [2]. A *model graph* H with $V_H = \{h_1, \dots, h_k\}$ has two types of edges: solid and dotted edges, and an H*-partition* of a graph G is a partition of V_G into k (nonempty) sets $V_1, \dots, V_k$ such that for all vertices $u \in V_i$, $v \in V_j$ and for all $1 \leq i < j \leq k$ the following two conditions hold. Firstly, if h_ih_j is a solid edge of H, then $uv \in E_G$. Secondly, if h_ih_j is a dotted edge of H, then $uv \notin E_G$. There are no restrictions when h_i and h_j are not adjacent. Let $2K_2$ be the model graph with vertices $h_1, \dots, h_4$ and two solid edges h_1h_3, h_2h_4, and $2S_2$ be the model graph with vertices $h_1, \dots, h_4$ and two dotted edges h_1h_3, h_2h_4. Then a graph G allows a surjective homomorphism to $\mathcal{C}_4$ if and only if G has a $2S_2$-partition if and only if its complement $\overline{G}$ has a $2K_2$-partition. The equivalent cases $H = 2K_2$ and $H = 2S_2$ are the only two cases of model graphs on at most four vertices that are still open. Especially $2K_2$-partitions have been well studied, see e.g. the recent papers of Dantas, Maffray and Silva [3] and Teixeira, Dantas and de Figueiredo [20]. The first paper [3] studies the $2K_2$-Partition problem for several graph classes and the second paper [20] even defines a new class of problems called $2K_2$-hard.

On the positive side, we can adapt our NP-hardness proof to determine the computational complexity of the H-Surjective Coloring problem when H is a cycle that is not loop-connected. Furthermore, the case when H is an odd irreflexive cycle can easily be shown to be NP-complete by a reduction from H-Coloring. Proofs are postponed to the journal version of our paper.

Acknowledgments. The authors would like to thank Barnaby Martin for fruitful discussions and useful comments on our paper.

References

1. Chen, J., Huang, X., Kanj, I.A., Xia, G.: Strong computational lower bounds via parameterized complexity. J. Comput. Syst. Sci. 72, 1346–1367 (2006)
2. Dantas, S., de Figueiredo, C.M., Gravier, S., Klein, S.: Finding H-partitions efficiently. RAIRO - Theoretical Informatics and Applications 39(1), 133–144 (2005)
3. Dantas, S., Maffray, F., Silva, A.: 2K2-partition of some classes of graphs. Discrete Applied Mathematics (to appear)
4. Dawar, A., Kreutzer, S.: Parameterized complexity of first-order logic, Electronic Colloquium on Computational Complexity, Report No. 131 (2009)
5. Diestel, R.: Graph theory. Graduate Texts in Mathematics, 3rd edn., vol. 173. Springer, Berlin
6. Downey, R.G., Fellows, M.R.: Parameterized complexity. Monographs in Computer Science. Springer, New York (1999)
7. Dvorak, Z., Král', D., Thomas, R.: Deciding first-order properties for sparse graphs. Technical report, Charles University, Prague (2009)
8. Feder, T., Hell, P.: List homomorphisms to reflexive graphs. J. Comb. Theory, Ser. B 72, 236–250 (1998)

9. Feder, T., Hell, P., Huang, J.: Bi-arc graphs and the complexity of list homomorphisms. Journal of Graph Theory 42, 61–80 (2003)
10. Feder, T., Hell, P., Jonsson, P., Krokhin, A., Nordh, G.: Retractions to pseudoforests. SIAM Journal on Discrete Mathematics 24, 101–112 (2010)
11. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: a study through datalog and group theory. SIAM Journal on Computing 28, 57–104 (1998)
12. Fiala, J., Kratochvíl, J.: Locally constrained graph homomorphisms – structure, complexity, and applications. Computer Science Review 2, 97–111 (2008)
13. Fiala, J., Paulusma, D.: A complete complexity classification of the role assignment problem. Theoretical Computer Science 349, 67–81 (2005)
14. Hell, P., Nešetřil, J.: On the complexity of H-colouring. Journal of Combinatorial Theory, Series B 48, 92–110 (1990)
15. Hell, P., Nešetřil, J.: Graphs and Homomorphisms. Oxford University Press, Oxford (2004)
16. Impagliazzo, R., Paturi, R., Zane, F.: Which problems have strongly exponential complexity? J. Comput. Syst. Sci. 63, 512–530 (2001)
17. Fleischner, H., Mujuni, E., Paulusma, D., Szeider, S.: Covering graphs with few complete bipartite subgraphs. Theoret. Comput. Sci. 410, 2045–2053 (2009)
18. Nešetřil, J., Ossona de Mendez, P.: On nowhere dense graphs. Technical report, Charles University, Prague (2008)
19. Patrignani, M., Pizzonia, M.: The Complexity of the Matching-Cut Problem. In: Brandstädt, W.A., Le, V.B. (eds.) WG 2001. LNCS, vol. 2204, pp. 284–295. Springer, Heidelberg (2001)
20. Teixeira, R.B., Dantas, S., de Figueiredo, C.M.H.: The external constraint 4 nonempty part sandwich problem. Discrete Applied Mathematics (to appear)
21. Vikas, N.: Computational complexity of compaction to reflexive cycles. SIAM Journal on Computing 32, 253–280 (2002)
22. Vikas, N.: Compaction, Retraction, and Constraint Satisfaction. SIAM Journal on Computing 33, 761–782 (2004)
23. Vikas, N.: A complete and equal computational complexity classification of compaction and retraction to all graphs with at most four vertices and some general results. J. Comput. Syst. Sci. 71, 406–439 (2005)

Compressed Membership in Automata with Compressed Labels

Markus Lohrey and Christian Mathissen

Institut für Informatik, Universität Leipzig, Germany
{lohrey,mathissen}@informatik.uni-leipzig.de

Abstract. The algorithmic problem of whether a compressed string is accepted by a (nondeterministic) finite state automaton with compressed transition labels is investigated. For string compression, straight-line programs (SLPs), i.e., context-free grammars that generate exactly one string, are used. Two algorithms for this problem are presented. The first one works in polynomial time, if all transition labels are nonperiodic strings (or more generally, the word length divided by the period is bounded polynomially in the input size). This answers a question of Plandowski and Rytter. The second (nondeterministic) algorithm is an NP-algorithm under the assumption that for each transition label the period is bounded polynomially in the input size. This generalizes the NP upper bound for the case of a unary alphabet, shown by Plandowski and Rytter.

1 Introduction

The topic of this paper is algorithms on compressed strings. The goal of such algorithms is to check properties of compressed strings and thereby beat a straightforward "decompress-and-check" strategy. Potential applications for such algorithms can be found for instance in genome databases, where massive volumes of string data are stored and analyzed. When talking about algorithms on compressed strings, one has to make precise the used compression scheme. Here, as in previous papers, we choose *straight-line programs* (SLPs); these are context-free grammars that generate exactly one string. Straight-line programs turned out to be a very flexible and mathematically clean compressed representation of strings. Several other dictionary-based compressed representations, like for instance Lempel-Ziv (LZ) factorizations [15], can be converted in polynomial time into SLPs and vice versa [13]. This implies that complexity results can be transferred from SLP-encoded input strings to LZ-encoded input strings.

Several algorithmic problems for SLP-compressed input strings were considered in the past, e.g. equivalence and pattern matching [5, 12], word problems for certain groups and monoids [8, 9, 11, 14], and membership problems for various language classes [3, 7, 9, 13]. In this paper, we study the membership problem for compressed words in automata with compressed labels. In this problem, the input consists of an SLP $\mathbb{A}$ and a nondeterministic automaton, where each transition is labeled with an SLP. Such an automaton generates a language in the obvious way, and it is asked whether the string generated by the SLP $\mathbb{A}$ belongs to that language. This problem was first studied in [13]; it is easily seen to be in PSPACE. Moreover, it was shown to be NP-complete for a unary alphabet in [13]. In fact, NP-hardness in the unary case follows directly from

NP-hardness of the SUBSETSUM problem. To the knowledge of the authors no better lower bound than NP-hardness is known for the non-unary case. This paper contains two algorithms for the membership problem for compressed words in automata with compressed labels (let $\mathcal{A}$ be the input automaton with compressed labels).

The first algorithm is deterministic and works in polynomial time if every SLP appearing in $\mathcal{A}$ generates a string with a small order (polynomial in the total input size). Here the order of a string is its length divided by its smallest period. Hence, having a small order means that the string looks quite aperiodic. In fact, as a corollary we obtain a polynomial time algorithm for the case that every SLP in $\mathcal{A}$ generates a nonperiodic word. This solves open problem 3 from [13]. The second algorithm is nondeterministic and works in polynomial time if every SLP in $\mathcal{A}$ generates a string with a small period (polynomial in the total input size). This generalizes the NP bound for the unary case from [13] (a unary word has period 1).

Hence, these two algorithms cover two extreme cases (almost nonperiodic versus highly periodic). But they do not cover the general case. An SLP $\mathbb{A}$ may generate a string, for which both the order and the period are exponential in the size of $\mathbb{A}$. Nevertheless, following [13], we conjecture that the general membership problem for compressed words in automata with compressed labels belongs to NP. We conclude this paper with another conjecture that implies the former one.

2 Preliminaries

We use $\mathsf{poly}(n_1, \ldots, n_k)$ as an abbreviation for $(n_1 + \cdots + n_k)^{O(1)}$. For $n \leq m$, we denote with $[n, m]$ the interval $\{n, n+1, \ldots, m\}$. An *arithmetic progression* is a set of natural numbers of the form $\{b + i \cdot p \mid i \in [0, \ell]\}$. This set can be represented succinctly by the triple (b, p, ℓ), where all three numbers are binary coded.

Let us fix a finite alphabet Σ. For a string $w \in \Sigma^*$ and $1 \leq i \leq |w|$ let $w[i]$ denote the i-th symbol in w. Moreover, for $1 \leq i, j \leq |w|$ let $w[i : j] = w[i] \cdots w[j]$ if $i \leq j$ and let $w[i : j] = \varepsilon$ if $i > j$. A number $1 \leq p \leq |w| - 1$ is a *period* of w if $w[i] = w[i + p]$ for all $1 \leq i \leq |w| - p$. With $\mathsf{per}(w)$ we denote the smallest period of w, where we set $\mathsf{per}(w) = |w|$ if w has no period. Let $\mathsf{ord}(w) = \lfloor \frac{|w|}{\mathsf{per}(w)} \rfloor$ be the *order* of w. Then $w = u^{\mathsf{ord}(w)} v$, where u is *primitive* (i.e., it is not of the form x^n for a string x and $n \geq 2$) and v is a proper prefix of u (possibly empty). A string w is called *nonperiodic*, if $\mathsf{ord}(w) = 1$. Two words $u, v \in \Sigma^*$ are *conjugated*, if there exist $x, y \in \Sigma^*$ with $u = xy$ and $v = yx$. An occurrence of a word p in another word t is a number $i \in [0, |t| - |p|]$ such that $w[i + 1 : i + |p|] = p$. We say that this occurrence i *covers* all positions from the interval $[i + 1, i + |p|]$, and that it *touches* all positions from the interval $[i, i + |p|]$.

Lemma 1 (cf. [5, Lemma 1]). *Let $t, p \in \Sigma^*$ and $j \in [0, |t|]$. The set of all occurrences of p in t that touch position j is an arithmetic progression of size at most* $\mathsf{ord}(p)$.

In Section 4.2 we need some basic concepts concerning *string rewriting systems*, see [1] for more details. A string rewriting system R over Σ is a finite subset of $\Sigma^* \times \Sigma^*$. A pair $(\ell, r) \in R$ is called a rule of R and is often written as $\ell \to r$. R defines a rewrite relation $\to_R$ as follows: $u \to_R v$ for $u, v \in \Sigma^*$ if there exist $x, y \in \Sigma^*$ and a rule $(\ell, r) \in R$

such that $u = x\ell y$ and $v = xry$. The system R is *terminating* if there does not exist an infinite chain $u_0 \to_R u_1 \to_R u_2 \to_R \cdots$. Moreover, R is called *confluent* (resp. *locally confluent*) if for all $u, v, w \in \Sigma^*$ such that $u \to_R^* v$ and $u \to_R^* w$ (resp. $u \to_R v$ and $u \to_R w$) there exists $x \in \Sigma^*$ with $v \to_R^* x$ and $w \to_R^* x$. By Newman's lemma, a terminating system is confluent if and only if it is locally confluent. Moreover, for terminating systems, local confluence is decidable. For this one has to consider *critical pairs* that result from overlapping left hand sides, see [1] for more details. Let us set $\mathsf{IRR}(R) = \Sigma^* \setminus \{u \mid \exists v : u \to_R v\}$. If R is terminating and confluent, then for every $u \in \Sigma^*$ there exists a unique $v \in \mathsf{IRR}(R)$ such that $u \to_R^* v$; it is called the *irreducible normal form* of u and is denoted by $\mathsf{NF}_R(u)$.

A *straight-line program (SLP)* over the terminal alphabet Σ is a context-free grammar $\mathbb{A} = (N, \Sigma, S, P)$ (N is the set of nonterminals, Σ is the set of terminals, $S \in N$ is the initial nonterminal, and $P \subseteq N \times (N \cup \Sigma)^*$ is the set of productions) such that: (i) for every $A \in N$ there exists exactly one production of the form $(A, \alpha) \in P$ for $\alpha \in (N \cup \Sigma)^*$, and (ii) the relation $\{(A, B) \in N \times N \mid (A, \alpha) \in P, B \text{ occurs in } \alpha\}$ is acyclic. The transitive closure of this relation is also called the *hierarchical order* of $\mathbb{A}$; it is a partial order. A production (A, α) is also written as $A \to \alpha$. Clearly, the language generated by the SLP $\mathbb{A}$ consists of exactly one word that is denoted by $\mathsf{val}(\mathbb{A})$. More generally, from every nonterminal $A \in N$ we can generate exactly one word that is denoted by $\mathsf{val}_{\mathbb{A}}(A)$ (thus $\mathsf{val}(\mathbb{A}) = \mathsf{val}_{\mathbb{A}}(S)$). We omit the index $\mathbb{A}$ if the underlying SLP is clear from the context. The size of $\mathbb{A}$ is $|\mathbb{A}| = \sum_{(A,\alpha)\in P} |\alpha|$. Every SLP $\mathbb{A}$ with $\mathsf{val}(\mathbb{A}) \neq \varepsilon$ can be transformed in polynomial time into an equivalent SLP in *Chomsky normal form*, i.e., all productions have the form (A, a) with $a \in \Sigma$ or (A, BC) with $B, C \in N$. In the sequel we assume that all SLPs are in Chomsky normal form.

Let us state some algorithmic problems that can be easily solved in polynomial time:

– Given an SLP $\mathbb{A}$, calculate $|\mathsf{val}(\mathbb{A})|$.
– Given an SLP $\mathbb{A}$ and a number $i \in \{1, \ldots, |\mathsf{val}(\mathbb{A})|\}$, calculate $\mathsf{val}(\mathbb{A})[i]$; this problem is in fact P-complete [6].

Let $\mathbb{A}$ be an SLP with a production (A, BC). An occurrence $i \in [0, |\mathsf{val}(A)| - |p|]$ of the word p in $\mathsf{val}(A)$ *touches the cut of* A, if this occurrence touches position $|\mathsf{val}(B)|$. The following result by Lifshits implies in particular, that for given SLPs $\mathbb{A}$ and $\mathbb{B}$ one can check in time $O(|\mathbb{A}| \cdot |\mathbb{B}|^2)$, whether $\mathsf{val}(\mathbb{A})$ occurs as a pattern in $\mathsf{val}(\mathbb{B})$.

Theorem 2 ([5]). *For two given SLPs $\mathbb{A}$ and $\mathbb{B}$ we can compute in time $O(|\mathbb{A}| \cdot |\mathbb{B}|^2)$ a table that contains for every nonterminal B of $\mathbb{B}$ an arithmetic progression (stored by three binary encoded numbers) for the set of all occurrences of $\mathsf{val}(\mathbb{A})$ in $\mathsf{val}_{\mathbb{B}}(B)$ that are touching the cut of B.*

An *automaton with compressed labels* is a tuple $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, Σ is a finite alphabet, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and δ is a finite set of transitions of the form $(p, \mathbb{A}, q)$, where p and q are states and $\mathbb{A}$ is an SLP over Σ, for which we assume $\mathsf{val}(\mathbb{A}) \neq \varepsilon$ (ε-transitions can be eliminated). A transition $(p, \mathbb{A}, q)$ with $|\mathsf{val}(\mathbb{A})| = 1$ is called *atomic*. The size of $\mathcal{A}$ is $|\mathcal{A}| = |Q| + \sum_{(p,\mathbb{A},q)\in\delta} |\mathbb{A}|$. We say that a word w labels a path from state p to state q in $\mathcal{A}$ if there exists a sequence of transitions $(p_0, \mathbb{A}_0, p_1), (p_1, \mathbb{A}_1, p_2), \ldots, (p_{n-1}, \mathbb{A}_{n-1}, p_n) \in \delta$ $(n \geq 0)$ such that $p_0 = p$, $p_n = q$, and $w = \mathsf{val}(\mathbb{A}_0) \cdots \mathsf{val}(\mathbb{A}_{n-1})$. We say the transition starting at position $\sum_{i=0}^{\ell-1} |\mathsf{val}(\mathbb{A}_i)|$ and ending at position $\sum_{i=0}^{\ell} |\mathsf{val}(\mathbb{A}_i)|$ is $(p_\ell, \mathbb{A}_\ell, p_{\ell+1})$. The language $L(\mathcal{A}) \subseteq \Sigma^*$ is the set of all words that label a path from the initial state q_0 to some final state $q_f \in F$. We set $\mathsf{ord}(\mathcal{A}) = \max\{\mathsf{ord}(\mathsf{val}(\mathbb{A})) \mid (p, \mathbb{A}, q) \in \delta\}$ and $\mathsf{per}(\mathcal{A}) = \max\{\mathsf{per}(\mathsf{val}(\mathbb{A})) \mid (p, \mathbb{A}, q) \in \delta\}$. Note that in general, both $\mathsf{ord}(\mathcal{A})$ and $\mathsf{per}(\mathcal{A})$ are exponential in $|\mathcal{A}|$.

3 A Deterministic Algorithm

The goal of this section is to prove the following theorem.

Theorem 3. *Given an automaton with compressed labels $\mathcal{A}$ and an SLP $\mathbb{B}$, we can check* $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$ *in time* $\mathsf{poly}\big(|\mathbb{B}|, |\mathcal{A}|, \mathsf{ord}(\mathcal{A})\big)$.

Proof. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$, $\mathbb{B} = (N, \Sigma, S, P)$, and let $\mathbb{A}_1, \ldots, \mathbb{A}_n$ be a list of all SLPs that occur as labels in $\mathcal{A}$. By Theorem 2, we can compute in time $O(\sum_{i=1}^{n}(|\mathbb{A}_i| \cdot |\mathbb{B}|^2)) \leq O(|\mathcal{A}| \cdot |\mathbb{B}|^2)$ a table that contains for every $i \in [1, n]$ and every nonterminal $B \in N$ an arithmetic progression $\mathsf{AP}(i, B)$ for the set of all occurrences of $\mathbb{A}_i$ in $\mathsf{val}_{\mathbb{B}}(B)$ that are touching the cut of B. Moreover, by Lemma 1, this arithmetic progression contains at most $\mathsf{ord}(\mathsf{val}(\mathbb{A}_i))$ many numbers. In total, we have at most $|\mathbb{B}| \cdot \sum_{i=1}^{n} \mathsf{ord}(\mathsf{val}(\mathbb{A}_i)) \leq |\mathbb{B}| \cdot |\mathcal{A}| \cdot \mathsf{ord}(\mathcal{A})$ many numbers.

We now define a context-free grammar G with empty terminal alphabet. The two major facts about G are:

- G can be computed in time $\mathsf{poly}\big(|\mathbb{B}|, |\mathcal{A}|, \mathsf{ord}(\mathcal{A})\big)$.
- $\varepsilon \in L(G)$ if and only if $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$.

Since the word problem for context-free grammars can be decided in polynomial time, these two facts imply the theorem. The grammar G is constructed by a fixpoint process, where we add more and more nonterminals. The set of nonterminals contains the start nonterminal S_G; all other nonterminals are 5-tuples of the form (p, k, B, ℓ, q), where $p, q \in Q$, $B \in N$, and $k, \ell \in [0, |\mathsf{val}(B)|]$ with $k + \ell \leq |\mathsf{val}(B)|$. The intuition is that this 5-tuple should be viewed as the following assertion: Let w be the word that results from $\mathsf{val}(B)$ by cutting off the prefix of length k and the suffix of length ℓ, i.e., $w = \mathsf{val}(B)[k+1 : |\mathsf{val}(B)| - \ell]$. Then in the automaton $\mathcal{A}$ there exists a path from state p to state q labeled with the word w.

For G's start nonterminal S_G we introduce all productions of the form

$$S_G \to (0, q_0, S, q_f, 0), \tag{1}$$

where $q_f \in F$ is a final state of $\mathcal{A}$. Now assume that at some point we have introduced a new nonterminal (k, p, B, q, ℓ). We distinguish 5 cases:

Case 1. $(B \to CD) \in P$ and $|\mathsf{val}(C)| \leq k < |\mathsf{val}(B)| - \ell$, see Figure 1. We introduce the production

$$(k, p, B, q, \ell) \to (k - |\mathsf{val}(C)|, p, D, q, \ell).$$

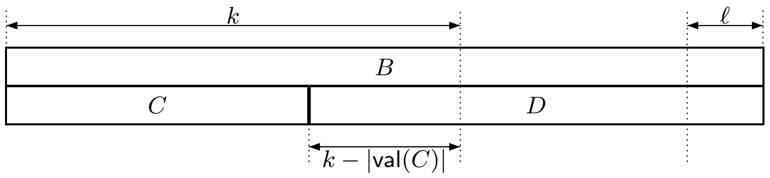


Fig. 1. Case 1

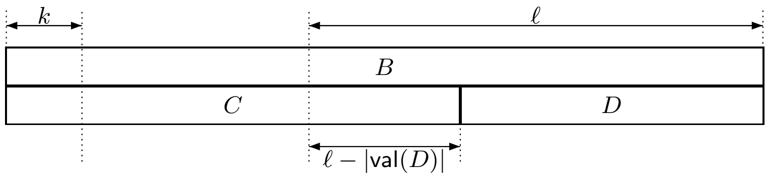


Fig. 2. Case 2

Case 2. $(B \to CD) \in P$ and $|\mathsf{val}(D)| \le \ell < |\mathsf{val}(B)| - k$, see Figure 2. We introduce the production

$$(k, p, B, q, \ell) \to (k, p, C, q, \ell - |\mathsf{val}(D)|).$$

Case 3. $(B \to CD) \in P$ and $k < |\mathsf{val}(C)|$, $\ell < |\mathsf{val}(D)|$, see Figure 3. We introduce a production for every transition $(r, \mathbb{A}_i, s)$ of $\mathcal{A}$ and every $j \in \mathsf{AP}(i, B)$ such that $j \ge k$ and $|\mathsf{val}(B)| - |\mathsf{val}(\mathbb{A}_i)| - j \ge \ell$. For such a choice, we introduce the production

$$(k, p, B, q, \ell) \to (k, p, C, r, |\mathsf{val}(C)| - j)\,(|\mathsf{val}(\mathbb{A}_i)| + j - |\mathsf{val}(C)|, s, D, q, \ell). \quad (2)$$

Case 4. $k + \ell = |\mathsf{val}(B)|$. If $p = q$, then we introduce the production

$$(k, p, B, q, \ell) \to \varepsilon.$$

Case 5. $(B \to a) \in P$ and $k = \ell = 0$. If in $\mathcal{A}$ there is a path from state p to state q labeled with the letter a, then we introduce the production

$$(0, p, B, q, 0) \to \varepsilon.$$

This concludes the description of G. It is straightforward to show that $\varepsilon \in L(G)$ if and only if $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$.

We claim that G contains at most $O\big(|\mathbb{B}|^7 \cdot |\mathcal{A}|^4 \cdot \mathsf{ord}(\mathcal{A})^2\big)$ many nonterminals. This clearly implies that G can be constructed in time $\mathsf{poly}\big(|\mathbb{B}|, |\mathcal{A}|, \mathsf{ord}(\mathcal{A})\big)$. For the second, third and fourth component of a G-nonterminal (except of S_G) there are in total $|Q|^2 \cdot |N| \le |\mathcal{A}|^2 \cdot |\mathbb{B}|$ possibilities. Let us bound the number of positions that may appear as a first component of a G-nonterminal (an analogous argument will apply to the fifth component). Let M_1 be the set of all positions that appear as a first component of a nonterminal of G. Moreover, let us define the set

$$J = \{|\mathsf{val}(\mathbb{A}_i)| + j - |\mathsf{val}(C)| \mid 1 \le i \le n, \exists B, D \in N : (B, CD) \in P, j \in \mathsf{AP}(i, B)\}.$$

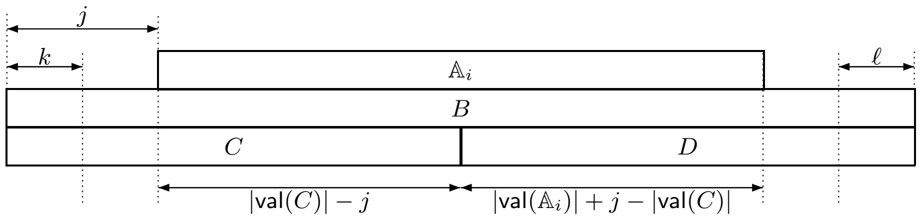


Fig. 3. Case 3

Every first component of the second G-nonterminal in the right-hand side of the G-production (2) is from J. Note that

$$|J| \le |\mathbb{B}| \cdot \sum_{i=1}^{n} \mathsf{ord}(\mathsf{val}(\mathbb{A}_i)) \le |\mathbb{B}| \cdot |\mathcal{A}| \cdot \mathsf{ord}(\mathcal{A}).$$

Let us now define a mapping f on $[0, |\mathsf{val}(\mathbb{B})|] \times N$ as follows:

$$f(k, B) = \begin{cases} (k - |\mathsf{val}(C)|, D) & \text{if } (B, CD) \in P, |\mathsf{val}(C)| \le k \\ (k, C) & \text{if } (B, CD) \in P, |\mathsf{val}(C)| > k \\ \text{undefined} & \text{otherwise} \end{cases}$$

This mapping f describes the way the first and third component of a G-nonterminal evolve when applying the productions from Case 1, 2, and 3 (for Case 3, we only consider the first G-nonterminal in the right-hand side of (2)). Note that for every $(k, B) \in [0, |\mathsf{val}(\mathbb{B})|] \times N$, there is $\alpha \le |N| - 1$ such that $f^{\alpha}(k, B) =$ undefined. Moreover, if $i \in M_1$, then there exists $(k, B) \in \{(0, S)\} \cup (J \times N)$ and $0 \le \alpha \le |N| - 1$ such that $f^{\alpha}(k, B) \in \{i\} \times N$. Hence, the size of M_1 is bounded by

$$(|J| \cdot |N| + 1) \cdot |N| \le (|\mathbb{B}|^2 \cdot |\mathcal{A}| \cdot \mathsf{ord}(\mathcal{A}) + 1) \cdot |\mathbb{B}| \in O(|\mathbb{B}|^3 \cdot |\mathcal{A}| \cdot \mathsf{ord}(\mathcal{A})).$$

Hence, the number of nonterminals of G can be bounded by $O\big(|\mathbb{B}|^7 |\mathcal{A}|^4 \mathsf{ord}(\mathcal{A})^2\big)$. This concludes the proof of the theorem. □

4 A Nondeterministic Algorithm

The goal of this section is to prove the following theorem:

Theorem 4. *Given an automaton with compressed labels $\mathcal{A}$ and an SLP $\mathbb{B}$, we can check* $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$ *nondeterministically in time* $\mathsf{poly}\big(|\mathbb{B}|, |\mathcal{A}|, \mathsf{per}(\mathcal{A})\big)$.

In a first step, we will deal with the special case that $\mathsf{per}(\mathcal{A}) = 1$, which means that every transition is labeled with a compressed unary word (Theorem 5 below). Note that the NP bound in Theorem 5 already generalizes [13, Theorem 4], since the unary alphabet for each transition is allowed to vary.

4.1 Compressed Unary Labels

Theorem 5. *Given an automaton $\mathcal{A}$ with compressed labels over unary alphabets and an SLP $\mathbb{B}$, we can check whether* $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$ *in* NP.

Proof. We give an algorithm for the case $\Sigma = \{0,1\}$. The general case is similar. W.l.o.g. we may assume that $\mathsf{val}(\mathbb{B}) \in 0\{0,1\}^*1$. Let $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ and $\mathbb{B} = (N, \{0,1\}, S, P)$.

Step 1. Let $m \geq 2, \beta_1, \ldots, \beta_m \geq 1$ such that $\mathsf{val}(\mathbb{B}) = 0^{\beta_1}1^{\beta_2}0^{\beta_3} \cdots 1^{\beta_m}$. Note that m might be exponentially big, however the size of the set $I = \{\beta_1, \ldots, \beta_m\}$ is bounded by the number of nonterminals of $\mathbb{B}$. The binary codings of the numbers β_i can be computed bottom-up. For each production $(A, BC) \in P$ we get a new number β_i in case $\mathsf{val}_{\mathbb{B}}(B)$ ends with the same symbol as $\mathsf{val}_{\mathbb{B}}(C)$ starts. We transform $\mathbb{B}$ into an SLP $\mathbb{C}$ over the alphabet $\Theta = \{X_i, Y_i \mid i \in I\}$ such that $\mathsf{val}(\mathbb{C}) = X_{\beta_1}Y_{\beta_2}X_{\beta_3} \cdots Y_{\beta_m}$. This can be done in deterministic polynomial time similar to the construction from [8, proof of Theorem 2]. The SLP $\mathbb{C}$ contains all nonterminals from V plus some auxiliary nonterminals. The right-hand side in $\mathbb{C}$ of an old variable $A \in V$ will be of the form Z or $Z_1A'Z_2$, where $Z, Z_1, Z_2 \in \Theta$ and A' is an auxiliary nonterminal. Consider a production $(A, BC) \in P$, and assume that the $\mathbb{C}$-productions $(B, Z_1B'Z_2), (C, Z_3C'Z_4)$ are already computed (the case that the right-hand side of B or C is a single symbol from Θ is similar). In case Z_2 is of the form X_i and Z_3 is of the form Y_j $(i, j \in I)$ or vice versa, we introduce the $\mathbb{C}$-productions (A, Z_1A', Z_4) and $(A', B'Z_2Z_3C')$. On the other hand, if, e.g., $Z_2 = X_i$ and $Z_3 = X_j$ $(i, j \in I)$, then we introduce the $\mathbb{C}$-productions $(A, Z_1A'Z_4)$ and $(A', B'X_{i+j}C')$.

Step 2. We build nondeterministically a new automaton $\mathcal{B} = (Q, \Theta, q_0, \delta', F)$ (with noncompressed labels). For all $q, p \in Q$ and for each $i \in I$ we guess whether there is a path in $\mathcal{A}$ from q to p labeled with 0^i (resp. 1^i). If this is true, then we add a transition (q, X_i, p) (resp. (q, Y_i, p)) to δ'.

Step 3. For each pair (q, p) it can be checked nondeterministically in time $\mathsf{poly}(|\mathbb{B}|, |\mathcal{A}|)$ whether there is a path from q to p in $\mathcal{A}$ with label 0^{β_i} (resp. 1^{β_i}) (see [13, Theorem 4]). So for each transition (q, X_i, p) and (q, Y_i, p) in $\mathcal{B}$ we can check whether there is in fact a corresponding path in $\mathcal{A}$.

Step 4. We can check deterministically in time $\mathsf{poly}(|\mathbb{C}|, |\mathcal{B}|)$ whether $\mathsf{val}(\mathbb{C}) \in L(\mathcal{B})$ (see [13, Theorem 2(a)]). Clearly, $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$ if and only if there is an automaton $\mathcal{B}$, obtained as described above, such that $\mathsf{val}(\mathbb{C}) \in L(\mathcal{B})$. □

In the rest of this paper, we will prove Theorem 4. First, we have to do some combinatorics on words.

4.2 Some Combinatorics on Words

The following lemma is well known.

Lemma 6 (e.g. [10]). *Let $u \in \Sigma^*$ be primitive and $u^2 = vuw$ for some $v, w \in \Sigma^*$. Then either $v = \varepsilon$ or $w = \varepsilon$.*

The next lemma is an easy consequence of the well-known periodicity theorem of Fine and Wilf.

Lemma 7 (cf. [2, Corollary 6.2]). *Let $u \neq v$ be two primitive words that are not conjugate, and let $n, m \in \mathbb{N}$. Then u^n and v^m do not have a common factor of length $|u| + |v|$.*

Let now $U = \{u_1, \dots, u_n\} \subseteq \Sigma^+$ be a collection of primitive words that are pairwise not conjugated. Later U will consist of the primitive roots of labels occurring in an automaton with compressed labels. For $1 \leq i \leq n$ let $\alpha_i = 1 + \lceil |v|/|u_i| \rceil \geq 2$, where v is a longest word in U. Lemma 7 implies:

Lemma 8. *For $i \neq j$, $u_i^{\alpha_i}$ is not a factor of a word from u_j^*.*

Let $X_1, \dots, X_n$ be fresh letters which are not in Σ. We now define a string rewriting system R_U over the alphabet $\Sigma \cup \{X_1, \dots, X_n\}$. First, for $1 \leq i \leq n$ let R_i consist of the following 4 rules:

$$u_i^{2\alpha_i+1} \to u_i^{\alpha_i} X_i u_i^{\alpha_i} \tag{3}$$
$$u_i^{\alpha_i+1} X_i \to u_i^{\alpha_i} X_i^2 \tag{4}$$
$$X_i u_i^{\alpha_i+1} \to X_i^2 u_i^{\alpha_i} \tag{5}$$
$$X_i u_i^{\alpha_i} X_i \to X_i^{\alpha_i+2} \tag{6}$$

Finally, let $R_U = \bigcup_{i=1}^n R_i$. Let m_U be the maximal length of a left-hand side of R_U. The following obvious fact is useful in the further investigations:

Fact 9. *If $u \to_{R_U}^* v$, then u can be obtained from v by replacing some (but not necessarily all) occurrences of X_i by u_i ($1 \leq i \leq n$).*

Clearly, R_U is terminating. Moreover, we have:

Lemma 10. *R_U is confluent.*

Proof. No left-hand side of R_U is a factor of another left-hand side. Hence, we have to check critical pairs that result from overlappings between left-hand sides. Rule (3) replaces an occurrence of u_i by X_i within the context $(u_i^{\alpha_i}, u_i^{\alpha_i})$. Similarly, (4) (resp., (5)) replaces an occurrence of u_i by X_i within the context $(u_i^{\alpha_i}, X_i)$ (resp. $(X_i, u_i^{\alpha_i})$). Finally, (6) replaces an occurrence of $u_i^{\alpha_i}$ by $X_i^{\alpha_i}$ within the context (X_i, X_i). These observations imply that critical pairs that result from an overlapping between a left-hand side of R_i and a left-hand side of R_j with $i \neq j$ can be directly resolved: Lemma 8 implies that the replaced parts in the left-hand sides cannot overlap, i.e., the overlapping is restricted to the context. It remains to consider overlappings between left-hand sides of some R_i. Again, those overlappings that are restricted to the context can be directly resolved. Since u_i does not occur properly in u_i^2 (Lemma 6), the critical pairs from Figure 4 (shown together with the resolving derivations) remain (arrows are labeled with the applied rule and possibly a number indicating the number of rule applications). This concludes the confluence proof. □

$$\begin{tikzcd}
& u_i^{2\alpha_i+1}X_i \arrow[dl, "(3)"'] \arrow[dr, "(4)"] & \\
u_i^{\alpha_i}X_iu_i^{\alpha_i}X_i \arrow[dr, "(6)"'] & & u_i^{2\alpha_i}X_i^2 \arrow[dl, "\alpha_i"', "(4)"] \\
& u_i^{\alpha_i}X_i^{\alpha_i+2} &
\end{tikzcd}
\qquad
\begin{tikzcd}
& X_iu_i^{2\alpha_i+1} \arrow[dl, "(3)"'] \arrow[dr, "(5)"] & \\
X_iu_i^{\alpha_i}X_iu_i^{\alpha_i} \arrow[dr, "(6)"'] & & X_i^2u_i^{2\alpha_i} \arrow[dl, "\alpha_i"', "(5)"] \\
& X_i^{\alpha_i+2}u_i^{\alpha_i} &
\end{tikzcd}$$

$$\begin{tikzcd}
& u_i^{2\alpha_i+1+k} \arrow[dl, "(3)"'] \arrow[dr, "(3)"] & \\
u_i^{\alpha_i}X_iu_i^{\alpha_i+k} \arrow[dr, "k", "(5)"'] & & u_i^{\alpha_i+k}X_iu_i^{\alpha_i} \arrow[dl, "k"', "(4)"] \\
& u_i^{\alpha_i}X_i^{k+1}u_i^{\alpha_i} &
\end{tikzcd}
\qquad
\begin{tikzcd}
& X_iu_i^{\alpha_i+1}X_i \arrow[dl, "(4)"'] \arrow[dr, "(5)"] & \\
X_iu_i^{\alpha_i}X_i^2 \arrow[dr, "(6)"'] & & X_i^2u_i^{\alpha_i}X_i \arrow[dl, "(6)"] \\
& X_i^{\alpha_i+3} &
\end{tikzcd}$$

Fig. 4. Proving confluence of R_U

In the following, we write NF_U for NF_{R_U}. The next lemma is needed in order to prove the crucial Lemma 12 below.

Lemma 11. *Assume that $x, y \in (\Sigma \cup \{X_1, \ldots, X_n\})^*$, $xu_i^{\alpha_i+1} \in \mathsf{IRR}(R_U)$, and $y \neq \varepsilon$ neither starts with u_i nor X_i. If $xu_i^{\alpha_i+1}y \to_{R_U}^* v$, then $v = xu_i^{\alpha_i+1}z$ for some $z \neq \varepsilon$ that neither starts with u_i nor X_i.*

Proof. Using induction, it suffices to prove the lemma for the case that the derivation $xu_i^{\alpha_i+1}y \to_{R_U}^* v$ has length one, i.e., $xu_i^{\alpha_i+1}y \to_{R_U} v$. The case that v is obtained from $xu_i^{\alpha_i+1}y$ by applying a rule within the suffix y (i.e., $y \to_{R_U} y'$ and $v = xu_i^{\alpha_i+1}y'$) is clear; one can use Fact 9 to see that y' neither starts with u_i nor X_i. So, we have to consider an occurrence of a left-hand side ℓ of R_U that starts in $xu_i^{\alpha_i+1}$ and ends in y. Assume that ℓ is a left-hand side of $R_j \subseteq R_U$. If $i \neq j$, then Lemma 8 implies that the occurrence of ℓ has to start in the suffix $u_i^{\alpha_i+1}$ of $xu_i^{\alpha_i+1}$. So, only the rules $u_j^{2\alpha_j+1} \to u_j^{\alpha_j}X_ju_j^{\alpha_j}$ and $u_j^{\alpha_j+1}X_j \to u_j^{\alpha_j}X_j^2$ have to be considered. By Lemma 8, the length of an overlapping between $u_i^{\alpha_i+1}$ and $u_j^{2\alpha_j+1}$ (resp., $u_j^{\alpha_j+1}X_j$) is bounded by $\alpha_j \cdot |u_j|$. Hence, the prefix $xu_i^{\alpha_i+1}$ is not modified in the rewrite step. Moreover, the rewrite step either does not modify the first $|u_i|$ many positions of y or produces an occurrence of X_j within one of the first $|u_i|$ many positions of y. Hence, indeed, $v = xu_i^{\alpha_i+1}z$ for some $z \neq \varepsilon$ that neither starts with u_i nor X_i. Finally, consider the case $i = j$. By Lemma 6 the occurrence of the left-hand side ℓ of R_i has to start in one of the last $|u_i|$ many positions of $xu_i^{\alpha_i+1}$ (otherwise y would start with u_i or X_i). But then, the prefix $xu_i^{\alpha_i+1}$ as well as the first $|u_i|$ many positions of y are not modified in the rewrite step. $\square$

Lemma 12. *Let $s, t \in \mathsf{IRR}(R_U)$, $s = s_1s_2$, and $t = t_1t_2$ with ($|s_2| = m_U$ or $s = s_2$) and ($|t_1| = m_U$ or $t = t_1$). Then, $\mathsf{NF}_U(st) = s_1\mathsf{NF}_U(s_2t_1)t_2$.*

Proof. We only consider the case $|s_2| = |t_1| = m_U$. Since $st \to^*_{R_U} s_1 \mathsf{NF}_U(s_2t_1)t_2$, it suffices to show that $s_1\mathsf{NF}_U(s_2t_1)t_2 \in \mathsf{IRR}(R_U)$. Assume for contradiction that $s_1\mathsf{NF}_U(s_2t_1)t_2$ is reducible. Since, $s_1, \mathsf{NF}_U(s_2t_1), t_2 \in \mathsf{IRR}(R_U)$, there has to be an occurrence of a left-hand side ℓ that starts in the prefix s_1 or that ends in the suffix t_2. By symmetry assume the former. Hence, $\ell = \ell_1\ell_2$ with $\ell_1 \neq \varepsilon \neq \ell_2$, ℓ_1 is a suffix of s_1, and ℓ_2 is a prefix of $\mathsf{NF}_U(s_2t_1)t_2$. We distinguish the following cases:

Case 1. $\ell = u_i^{2\alpha_i+1}$ for some $1 \le i \le n$. Then by Fact 9, $\ell_2 \in \Sigma^+$ must be a prefix of s_2t as well. Since $|s_2| \ge (2\alpha_i + 1)\cdot|u_i|$, it follows that ℓ_2 is in fact a prefix of s_2. Hence $s = s_1s_2$ is reducible, a contradiction.

Case 2. $\ell = X_iu_i^{\alpha_i+1}$ for some $1 \le i \le n$. Since again $\ell_2 \in \Sigma^+$, we can argue as in Case 1.

Case 3. $\ell = u_i^{\alpha_i+1}X_i$. Let $\ell_1 = u_i^{m_1}u'$ (it is a suffix of s_1) and $\ell_2 = u''u_i^{m_2}X_i$ with $u_i = u'u''$ and $\alpha_i + 1 = m_1 + 1 + m_2$. Then $u''u_i^{m_2}$ is a prefix of s_2t. Note that $|s_2| \ge (2\alpha_i + 1)\cdot|u_i|$. Let $m_3 \ge m_2$ maximal such that $u''u_i^{m_3}$ is a prefix of s_2t. We must have $m_1+1+m_3 < 2\alpha_i+1$, because otherwise $s = s_1s_2$ would contain an occurrence of $u_i^{2\alpha_i+1}$ and therefore would be reducible. Since $|s_2| \ge (2\alpha_i + 1)\cdot|u_i|$, $u''u_i^{m_3}$ must be a prefix of s_2. Let $s_1 = xu_i^{m_1}u'$ and $s_2 = u''u_i^{m_3}y$. Since $m_3 + 1 < 2\alpha_i + 1$ and $|s_2| \ge (2\alpha_i + 1)\cdot|u_i|$, we have $|y| \ge |u_i|$ Now, consider the word $xu_i^{m_1+1+m_3}yt_1 = s_1s_2t_1$. We claim that yt_1 does not start with u_i or X_i. If it would do so, then, since $|y| \ge |u_i|$, y would start with u_i or X_i. This contradicts either the maximality of m_3 (if y starts with u_i) or implies that $s = s_1s_2$ contains an occurrence of $u_i^{\alpha_i+1}X_i$ (if y starts with X_i) and is therefore reducible. Hence yt_1 neither starts with u_i nor X_i. We can therefore apply Lemma 11 to $xu_i^{m_1+1+m_3}yt_1 = s_1s_2t_1 \to^*_{R_U} s_1\mathsf{NF}_U(s_2t_1)$. It follows that $s_1\mathsf{NF}_U(s_2t_1)$ has the form $xu_i^{m_1+1+m_3}z$, where $z \neq \varepsilon$ neither starts with u_i nor X_i. But by our assumption $s_1\mathsf{NF}_U(s_2t_1)t_2$ starts with $x\ell = xu_i^{m_1+1+m_2}X_i$. This leads to a contradiction, since $m_3 \ge m_2$.

Case 4. $\ell = X_iu_i^{\alpha_i}X_i$. Can be shown analogously to Case 3. □

Lemma 12 allows us to prove Lemma 13 below. For this, an extension of SLPs is useful. A *composition system* $\mathbb{B} = (N, \Sigma, S, P)$ is defined analogously to an SLP, but in addition to productions of the form $A \to \alpha$ ($A \in N, \alpha \in (N \cup \Sigma)^*$) it may also contain productions of the form $A \to B[i : j]$ for $N \in V$ and $i, j \in \mathbb{N}$. For such a production we define $\mathsf{val}_{\mathbb{B}}(A) = \mathsf{val}_{\mathbb{B}}(B)[i : j]$. The size of this production is $1 + \lceil\log_2(i)\rceil + \lceil\log_2(j)\rceil$. As for SLPs we define $\mathsf{val}(\mathbb{B}) = \mathsf{val}_{\mathbb{B}}(S)$. In [4], Hagenah presented a polynomial time algorithm, which transforms a given composition system $\mathbb{B}$ into an SLP $\mathbb{C}$ such that $\mathsf{val}(\mathbb{C}) = \mathsf{val}(\mathbb{B})$.[1] Below, we allow more general kinds of productions, where right-hand sides are arbitrary words, built up from terminals, nonterminals and symbols $B[i : j]$ for a nonterminal B and $i, j \in \mathbb{N}$. The semantics of such productions is the obvious one. Clearly, productions of this more general form can be transformed in polynomial time into the above standard form.

Lemma 13. *From a given SLP* $\mathbb{A} = (N, \Sigma, S, P)$ *and a set* U *as above, we can compute in time* $\mathsf{poly}(\sum_{i=1}^n |u_i|, |\mathbb{A}|)$ *an SLP* $\mathbb{B}$ *such that* $\mathsf{val}(\mathbb{B}) = \mathsf{NF}_U(\mathsf{val}(\mathbb{A}))$.

[1] The thesis [4] is written in German. An english presentation of Hagenah's algorithm can be found in [14].

Proof. Using Hagenah's algorithm, it suffices to construct in polynomial time a composition system $\mathbb{B} = (N, \Sigma \cup \{X_1, \ldots, X_n\}, S, R)$ such that $\mathsf{val}(\mathbb{B}) = \mathsf{NF}_U(\mathsf{val}(\mathbb{A}))$. To this aim we successively add productions to $\mathbb{B}$. W.l.o.g. assume that $\mathbb{A}$ is in Chomsky normal form. First, we put all productions $(A \to a) \in P$ with $a \in \Sigma$ into R. Now, consider a production $(A \to BC) \in P$, and assume that $\mathbb{B}$ contains already enough productions so that $\mathsf{val}_{\mathbb{B}}(B) = \mathsf{NF}_U(\mathsf{val}_{\mathbb{A}}(B))$ and $\mathsf{val}_{\mathbb{B}}(C) = \mathsf{NF}_U(\mathsf{val}_{\mathbb{A}}(C))$. Let $k_B = |\mathsf{val}_{\mathbb{B}}(B)|$ and $k_C = |\mathsf{val}_{\mathbb{B}}(C)|$, these numbers can be computed in time $\mathsf{poly}(|\mathbb{A}|)$. Moreover, in time $\mathsf{poly}(\sum_{i=1}^{n} |u_i|, |\mathbb{A}|)$, we can compute the words $x = \mathsf{val}_{\mathbb{B}}(B)[k_B - m_U + 1 : k_B]$, $y = \mathsf{val}_{\mathbb{B}}(C)[1 : m_U]$, and $z = \mathsf{NF}_U(xy)$. By Lemma 12, we have

$$\mathsf{NF}_U(\mathsf{val}_{\mathbb{A}}(A)) = \mathsf{val}_{\mathbb{B}}(B)[1 : k_B - m_U]\, z\, \mathsf{val}_{\mathbb{B}}(C)[m_U + 1 : k_C].$$

Hence, we introduce the production $A \to B[1 : k_B - m_U]\, z\, C[m_U + 1 : k_C]$. This concludes the construction of the composition system $\mathbb{B}$. $\square$

4.3 Proof of Theorem 4

Assume that $\mathcal{A}$ is an automaton with compressed labels. First we will transform $\mathcal{A}$ in time $\mathsf{poly}(|\mathcal{A}|, \mathsf{per}(\mathcal{A}))$ into an equivalent automaton with compressed labels with some additional nice properties. For an SLP $\mathbb{A}$ let us write $\mathsf{ord}(\mathbb{A})$ and $\mathsf{per}(\mathbb{A})$ for $\mathsf{ord}(\mathsf{val}(\mathbb{A}))$ and $\mathsf{per}(\mathsf{val}(\mathbb{A}))$, respectively, in the following. For simplicity, we will denote the automaton resulting from each of Steps 1–3 below again with $\mathcal{A}$.

Step 1. For each $\mathcal{A}$-transition $(p, \mathbb{A}, q)$, we can compute in time $\mathsf{poly}(|\mathbb{A}|)$ SLPs $\mathbb{U}$ and $\mathbb{V}$ such that $|\mathsf{val}(\mathbb{V})| < |\mathsf{val}(\mathbb{U})| = \mathsf{per}(\mathbb{A})$ and $\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{U})^{\mathsf{ord}(\mathbb{A})}\mathsf{val}(\mathbb{V})$ (see e.g. [3]). Moreover, in time $O(\mathsf{per}(\mathcal{A}))$, we can explicitly compute $u = \mathsf{val}(\mathbb{U})$ (it is primitive) and $v = \mathsf{val}(\mathbb{V})$. We now replace the transition $(p, \mathbb{A}, q)$ by a path of $|v| + 1$ many transitions: a transition labeled with an SLP for $u^{\mathsf{ord}(\mathbb{A})}$, followed by a sequence of $|v|$ atomic transitions, which give an v-labeled path ending in state q. Hence, we can assume that for every transition $(p, \mathbb{A}, q)$ of $\mathcal{A}$ we have $\mathsf{val}(\mathbb{A}) = u^n$ for a primitive word u. In the following, a transition $(p, \mathbb{A}, q)$ with $\mathsf{val}(\mathbb{A}) = u^n$ (u primitive) is just written as (p, u, n, q) (an atomic transition (p, a, q) can be viewed as $(p, a, 1, q)$). In fact, instead of an SLP for u^n, we can store the pair (u, n), where n is binary coded. All following steps are polynomial w.r.t. this new representation.

Step 2. Next, assume that there are two transitions (p, u, n, q) and (r, v, m, s) such that the primitive words u and v are conjugated. Hence, there are non-empty words $x, y \in \Sigma^+$ such that $u = xy$ and $v = yx$. We may assume that $m \geq 2$, as otherwise we replace the transition (r, v, m, s) by a path of atomic transitions. We now replace the transition (r, v, m, s) by a path of $|v| + 1$ many transitions: a path of $|y|$ many atomic transitions labeled y, followed by a transition labeled with the pair $(u, m - 1)$, followed by a path of $|x|$ many atomic transitions labeled with x.

Let $U = \{u_1, \ldots, u_n\}$ be the set of primitive words that occur in transitions of $\mathcal{A}$. W.l.o.g. we can assume that $\Sigma \subseteq U$. By Step 2, u_i and u_j are not conjugated for $i \neq j$. This allows us to construct the confluent and terminating system R_U from Section 4.2.

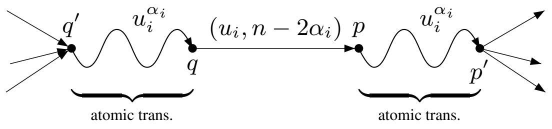


Fig. 5.

Let v be a longest word in U. Recall that we defined $\alpha_i = 1 + \lceil |v|/|u_i| \rceil$ for $1 \leq i \leq n$. We can compute all these numbers (even in unary notation) within our preprocessing time bound $\mathsf{poly}(|\mathcal{A}|, \mathsf{per}(\mathcal{A}))$.

Step 3. The aim of this step is to ensure the following technical condition.

Each transition (q, u_i, n, p) of $\mathcal{A}$ is either atomic or there are states q', p' such that: In $\mathcal{A}$ there is a path π of atomic transitions from q' to q labeled with $u_i^{\alpha_i}$ and there is a path π' of atomic transitions from p to p' labeled with $u_i^{\alpha_i}$. Moreover, on the path π the only state with indegree > 1 could be q' and on the path π' the only state with outdegree > 1 could be p'.

To ensure this condition, we first split each transition (q', u_i, n, p') with $n \leq 2\alpha_i$ into a path of atomic transitions. After that, each transition (q', u_i, n, p') with $n > 2\alpha_i$ is replaced by (see Figure 5):

- a path of atomic transitions from q' to some fresh state q labeled $u_i^{\alpha_i}$,
- a transition $(q, u_i, n - 2\alpha_i, p)$ for some fresh state p and
- a path of atomic transitions from p to p' labeled $u_i^{\alpha_i}$.

Observe that all our modifications preserve $L(\mathcal{A})$ and that they can be executed within the time bound $\mathsf{poly}(|\mathcal{A}|, \mathsf{per}(\mathcal{A}))$. This ends the preprocessing of the automaton $\mathcal{A}$.

Step 4. Let us introduce a new symbol X_i for every primitive word u_i (see also the definition of R_U). We now modify the automaton $\mathcal{A}$ as follows. For each primitive word $u_i \in U$ we replace every non-atomic transition (p, u_i, m, q) by (p, X_i, m, q). Moreover for any two states p, q of $\mathcal{A}$ we test whether there is a path of atomic transitions in $\mathcal{A}$ from p to q labeled u_i. If there is such a path we introduce a new transition (p, X_i, q). Let $\mathcal{B}$ denote our modified automaton. Now, consider an SLP $\mathbb{C}$ with $\mathsf{val}(\mathbb{C}) = \mathsf{NF}_U(\mathsf{val}(\mathbb{B}))$; such an SLP can be computed in polynomial time from $\mathbb{B}$ by Lemma 13. We claim that $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$ if and only if $\mathsf{val}(\mathbb{C}) \in L(\mathcal{B})$. This concludes the proof of Theorem 4 as the latter question belongs to NP by Theorem 5. So it remains to prove that indeed $\mathsf{val}(\mathbb{B}) \in L(\mathcal{A})$ if and only if $\mathsf{val}(\mathbb{C}) \in L(\mathcal{B})$.

For the if-direction, consider a path from the initial state q_0 to some final state q_f in $\mathcal{B}$ labeled $\mathsf{val}(\mathbb{C})$. Replacing every transition (q, X_i, m, p) by (q, u_i, m, p) and replacing every atomic transition (q, X_i, p) by an appropriate u_i-labeled path of atomic transitions in $\mathcal{A}$ gives a path in $\mathcal{A}$ from q_0 to q_f labeled $\mathsf{val}(\mathbb{B})$.

For the other direction, consider a path π from q_0 to some final state q_f in $\mathcal{A}$ labeled $\mathsf{val}(\mathbb{B})$ and fix an occurrence of $u_i^{\alpha_i} X_i^{\beta} u_i^{\alpha_i}$ in $\mathsf{val}(\mathbb{C}) = \mathsf{NF}_U(\mathsf{val}(\mathbb{B}))$ for some $u_i \in U$

($\beta > 0$). Let $j > 0$ be the position of $\mathsf{val}(\mathbb{B})$ such that the factor u_i^β corresponding to the block X_i^β occurs at j, i.e.,

$$\mathsf{val}(\mathbb{B})[j - \alpha_i|u_i| + 1 : j + \beta|u_i|] = u_i^{\alpha_i+\beta}.$$

Let (p, u_s, m, q) be the unique transition in π that starts at $k < j$ and ends at $\ell \geq j$. Thus, $\mathsf{val}(\mathbb{B})[k + 1 : \ell] = u_s^m$, i.e., k is an occurrence of u_s^m in $\mathsf{val}(\mathbb{B})$. Assume for contradiction that $\ell > j$. Hence (p, u_s, m, q) is non-atomic and by the condition from Step 3 above, the occurrence k of u_s^m in $\mathsf{val}(\mathbb{B})$ is preceded by $u_s^{\alpha_s}$, i.e.,

$$\mathsf{val}(\mathbb{B})[k - \alpha_s|u_s| + 1 : \ell] = u_s^{\alpha_s+m}.$$

If $s = i$, then $k < j < \ell$ implies that the occurrence $j - \alpha_i|u_i|$ of $u_i^{\alpha_i}$ (which covers all positions from $[j - \alpha_i|u_i| + 1, j]$) is strictly contained in the occurrence $k - \alpha_i|u_i|$ of $u_i^{\alpha_i+m}$ (which covers all positions from $[k-\alpha_i|u_i|+1, \ell]$). In particular, the occurrence $j - \alpha_i|u_i|$ of u_i is contained in an occurrence $< j - \alpha_i|u_i|$ of u_i^2. Lemma 6 implies that $j - \alpha_i|u_i| - |u_i| = j - (\alpha_i + 1)|u_i|$ is an occurrence of u_i as well. Hence, we have $\mathsf{val}(\mathbb{B})[j - (\alpha_i + 1)|u_i| + 1 : j] = u_i^{\alpha_i+1}$. But then, in $\mathsf{val}(\mathbb{C}) = \mathsf{NF}_U(\mathsf{val}(\mathbb{B}))$ we would obtain the factor $u_i^{\alpha_i} X_i^\gamma u_i^{\alpha_i}$ for some $\gamma > \beta$ instead of $u_i^{\alpha_i} X_i^\beta u_i^{\alpha_i}$, which is a contradiction. Hence $s \neq i$. But then either $u_i^{\alpha_i}$ is contained in $u_s^{\alpha_s+m}$ (if $k - \alpha_s|u_s| \leq j - \alpha_i|u_i|$) or $u_s^{\alpha_s}$ is contained in $u_i^{\alpha_i}$ (if $j - \alpha_i|u_i| \leq k - \alpha_s|u_s|$). This contradicts Lemma 8.

Hence $\ell = j$, i.e., there is a transition in π starting at j. A symmetric argument shows that there is a transition ending at $j + \beta|u_i|$ and hence there is a subpath π' of π from p' to q' that corresponds exactly to the block X_i^β. In fact, our argument also shows that this subpath π' cannot contain a non-atomic transition (p'', u_s, m, q'') with $s \neq i$ (we would obtain again a contradiction to Lemma 8). Hence, by Lemma 6 (u_i is primitive), π' can be decomposed into atomic paths labeled u_i and non-atomic transitions of the form (p, u_i, m, q). Hence, π' has a corresponding X_i^β-labeled path from p' to q' in $\mathcal{B}$. By doing this argument for all factors $u_i^{\alpha_i} X_i^\beta u_i^{\alpha_i}$ in $\mathsf{val}(\mathbb{C})$, we obtain a path from q_0 to q_f in $\mathcal{B}$ labeled $\mathsf{val}(\mathbb{C})$. This concludes the proof of Theorem 4.

5 Conclusion

We have considered the membership problem for a compressed string and an automaton with compressed labels. Two algorithms for this problem were developed. The first algorithm is deterministic and works in polynomial time if all transition labels have a small order (polynomial in the input size). The second algorithm is nondeterministic and works in polynomial time if all transition labels have a small period (polynomial in the input size), i.e., are highly periodic. Hence, these two algorithms cover two extreme cases (almost nonperiodic versus highly periodic). But the complexity of the general case remains open. Following [13], we conjecture that the general membership problem for compressed strings and automata with compressed labels belongs to NP. This would follow from the truth of the following conjecture:

Conjecture 14. If $\mathsf{val}(\mathbb{A}) \in L(\mathcal{A})$ for an SLP $\mathbb{A}$ and an automaton $\mathcal{A}$ with compressed labels, then there exists an accepting run of $\mathcal{A}$ on $\mathsf{val}(\mathbb{A})$ (viewed as a word over the set of transition triples of $\mathcal{A}$), which can be generated by an SLP of size $\mathsf{poly}(|\mathbb{A}|, |\mathcal{A}|)$.

Indeed, if this conjecture is true, we simply can guess an SLP $\mathbb{R}$ of size $\mathsf{poly}(|\mathbb{A}|, |\mathcal{A}|)$ over the set of transition tuples of $\mathcal{A}$. In polynomial time, we can check, whether $\mathbb{R}$ indeed generates an accepting run of $\mathcal{A}$ for some word (this is a regular property). Moreover, an SLP $\mathbb{B}$ for that word can be computed easily from $\mathbb{R}$. It remains to check whether $\mathsf{val}(\mathbb{A}) = \mathsf{val}(\mathbb{B})$, which can be done in polynomial time [12]. One might first study Conjecture 14 for the case that $\mathcal{A}$ is *deterministic*. Here, an automaton with compressed labels is deterministic, if for each pair of transition triples $(p, \mathbb{A}, q), (p, \mathbb{B}, r)$, neither $\mathsf{val}(\mathbb{A})$ is a prefix of $\mathsf{val}(\mathbb{B})$ nor vice versa. In this case, if there is an accepting run of $\mathcal{A}$ on a word w, there is a unique such run. Even for deterministic automata with compressed labels we are not aware of a better upper bound than PSPACE.

References

1. Book, R.V., Otto, F.: String–Rewriting Systems. Springer, Heidelberg (1993)
2. Choffrut, C., Karhumäki, J.: Combinatorics on words. In: Rozenberg, G., Salomaa, A. (eds.) Word, Language, Grammar. Handbook of Formal Languages, vol. 1 ch. 6, pp. 329–438. Springer, Heidelberg (1997)
3. Gasieniec, L., Karpinski, M., Plandowski, W., Rytter, W.: Efficient algorithms for Lempel-Ziv encoding (extended abstract). In: Karlsson, R., Lingas, A. (eds.) SWAT 1996. LNCS, vol. 1097, pp. 392–403. Springer, Heidelberg (1996)
4. Hagenah, C.: Gleichungen mit regulären Randbedingungen über freien Gruppen. PhD thesis, University of Stuttgart, Institut für Informatik (2000)
5. Lifshits, Y.: Processing compressed texts: A tractability border. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 228–240. Springer, Heidelberg (2007)
6. Lifshits, Y., Lohrey, M.: Querying and embedding compressed texts. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 681–692. Springer, Heidelberg (2006)
7. Lohrey, M.: Compressed membership problems for regular expressions and hierarchical automata. Internat. J. Found. Comput. Sci. 21(5), 817–841 (2010)
8. Lohrey, M., Schleimer, S.: Efficient computation in groups via compression. In: Diekert, V., Volkov, M.V., Voronkov, A. (eds.) CSR 2007. LNCS, vol. 4649, pp. 249–258. Springer, Heidelberg (2007)
9. Lohrey, M.: Word problems and membership problems on compressed words. SIAM J. Comput. 35(5), 1210–1240 (2006)
10. Lothaire, M.: Combinatorics on Words. Encyclopedia of Mathematics and its Applications, vol. 17. Addison-Wesley, Reading (1983)
11. Macdonald, J.: Compressed words and automorphisms in fully residually free groups. Internat. J. Algebra Comput. 20(3), 343–355 (2010)
12. Plandowski, W.: Testing equivalence of morphisms on context-free languages. In: van Leeuwen, J. (ed.) ESA 1994. LNCS, vol. 855, pp. 460–470. Springer, Heidelberg (1994)
13. Plandowski, W., Rytter, W.: Complexity of language recognition problems for compressed words. In: Jewels are Forever, Contributions on Theoretical Computer Science in Honor of Arto Salomaa, pp. 262–272. Springer, Heidelberg (1999)
14. Schleimer, S.: Polynomial-time word problems. Comment. Math. Helv. 83(4), 741–765 (2008)
15. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory 23(3), 337–343 (1977)

Locally Decodable Codes

Sergey Yekhanin

Microsoft Research Silicon Valley
yekhanin@microsoft.com

Locally Decodable Codes (LDCs) are a special kind of error-correcting codes. Error-correcting codes are used to ensure reliable transmission of information over noisy channels as well as to ensure reliable storage of information on a medium that may be partially corrupted over time (or whose reading device is subject to errors). In both of these applications the message is typically partitioned into small blocks and then each block is encoded separately. Such encoding strategy allows efficient random-access retrieval of the information, since one needs to decode only the portion of data one is interested in. Unfortunately, this strategy yields very poor noise resilience, since in case even a single block (out of possibly tens of thousands) is completely corrupted some information is lost. In view of this limitation it would seem preferable to encode the whole message into a single codeword of an error-correcting code. Such solution clearly improves the robustness to noise, but is also hardly satisfactory, since one now needs to look at the whole codeword in order to recover any particular bit of the message (at least in the case when classical error-correcting codes are used). Such decoding complexity is prohibitive for modern massive data-sets.

Locally decodable codes are error-correcting codes that avoid the problem mentioned above by having extremely efficient *sublinear-time* decoding algorithms. More formally, an r-query locally decodable code C encodes k-symbol messages $\mathbf{x}$ in such a way that one can probabilistically recover any symbol $\mathbf{x}(i)$ of the message by querying only r symbols of the (possibly corrupted) codeword $C(\mathbf{x})$, where r can be as small as 2.

Hadamard code. The classical Hadamard code encoding k-bit messages to 2^k-bit codewords provides the simplest nontrivial example of locally decodable codes. In what follows, let $[k]$ denote the set $\{1, \ldots, k\}$. Every coordinate in the Hadamard code corresponds to one (of 2^k) subsets of $[k]$ and stores the XOR of the corresponding bits of the message $\mathbf{x}$. Let $\mathbf{y}$ be an (adversarially corrupted) encoding of $\mathbf{x}$. Given an index $i \in [k]$ and $\mathbf{y}$, the Hadamard decoder picks a set S in $[k]$ uniformly at random and outputs the XOR of the two coordinates of $\mathbf{y}$ corresponding to sets S and $S \triangle \{i\}$. (Here, $\triangle$ denotes the symmetric difference of sets such as $\{1,4,5\} \triangle \{4\} = \{1,5\}$, and $\{1,4,5\} \triangle \{2\} = \{1,2,4,5\}$). It is not difficult to verify that if $\mathbf{y}$ differs from the correct encoding of $\mathbf{x}$ in at most δ fraction of coordinates than with probability $1 - 2\delta$ both decoder's queries go to uncorrupted locations. In such case, the decoder correctly recovers the i-th bit of $\mathbf{x}$. The Hadamard code allows for a super-fast recovery of the message bits (such as, given a codeword corrupted in 0.1 fraction of coordinates, one is able to recover any bit of the message with probability 0.8 by reading only two codeword bits).

The main parameters of interest in locally decodable codes are the codeword length and the query complexity. The length of the code measures the amount of redundancy that is introduced into the message by the encoder. The query complexity counts the number of bits that need to be read from the (corrupted) codeword in order to recover a single bit of the message. Ideally, one would like to have both of these parameters as small as possible. One however can not minimize the length and the query complexity simultaneously. There is a trade-off. On one end of the spectrum we have LDCs with the codeword length close to the message length, decodable with somewhat large query complexity. Such codes are useful for data storage and transmission. On the other end we have LDCs where the query complexity is a small constant but the codeword length is large compared to the message length. Such codes find applications in computational complexity theory and cryptography. The true shape of the trade-off between the codeword length and the query complexity of LDCs is not known. Determining it is a major open problem.

Currently there are three known families of locally decodable codes: classical Reed Muller codes [MS], multiplicity codes [KSY11], and matching vector codes [Yek08, Efr09]. In this talk we give a high level review of each of these families. We focus on the main ideas underlying the codes and omit many details. A detailed survey of a large body of work on LDCs (including a detailed treatment of the constructions, lower bounds, and applications) can be found in [Yek10].

References

[Efr09] Efremenko, K.: 3-query locally decodable codes of subexponential length. In: 41st ACM Symposium on Theory of Computing (STOC), pp. 39–44 (2009)

[KSY11] Kopparty, S., Saraf, S., Yekhanin, S.: High-rate codes with sublinear-time decoding. In: 43nd ACM Symposium on Theory of Computing, STOC (2011)

[MS] MacWilliams, F.J., Sloane, N.J.A.: The Theory of Error Correcting Codes. North Holland, Amsterdam

[Yek08] Yekhanin, S.: Towards 3-query locally decodable codes of subexponential length. Journal of the ACM 55, 1–16 (2008)

[Yek10] Yekhanin, S.: Locally decodable codes. Foundations and Trends in Theoretical Computer Science (2010) (to appear)

Precedence Automata and Languages

Violetta Lonati[1], Dino Mandrioli[2], and Matteo Pradella[2]

[1] DSI - Università degli Studi di Milano, via Comelico 39/41, Milano, Italy
lonati@dsi.unimi.it

[2] DEI - Politecnico di Milano, via Ponzio 34/5, Milano, Italy
{dino.mandrioli,matteo.pradella}@polimi.it

Abstract. Operator precedence grammars define a classical Boolean and deterministic context-free family (called Floyd languages or FLs). FLs have been shown to strictly include the well-known visibly pushdown languages, and enjoy the same nice closure properties. We introduce here Floyd automata, an equivalent operational formalism for defining FLs. This also permits to extend the class to deal with infinite strings to perform for instance model checking.

Keywords: Operator precedence languages, Deterministic Context-Free languages, Omega languages, Pushdown automata.

1 Introduction

The history of formal language theory has always paired two main and complementary formalisms to define and process –not only formal– languages: grammars or syntaxes and abstract machines or automata. The power and the complementary benefits of these two formalisms are so evident and well-known that it is certainly superfluous to remind them here. Also universally known are the conceptual relevance and practical impact of the family of context-free languages and the corresponding grammars paired with pushdown automata.

Among the many subfamilies that have been introduced throughout the last decades with various goals, operator precedence grammars, herewith renamed Floyd grammars (FGs) in honor of their inventor [1], represent a pioneering model mainly aimed at deterministic –and therefore efficient– parsing. Visibly pushdown languages (VPLs) are a much more recent subfamily of (deterministic) context-free languages introduced in the seminal paper [2] with the goal of extending the typical closure properties of regular languages to larger families of languages accepted by infinite-state machines; a major practical result is the possibility of extending such powerful verification technique as model checking beyond the scope of finite state machines. Along the usual tradition, VPLs have been characterized both in terms of abstract machines, the visibly pushdown automata (VPAs), and by means of a suitable subclass of context-free grammars.

Rather surprisingly, instead, investigation of the basic –and nice, indeed– properties of FGs has been suspended, probably as a consequence of the advent of other, more general, parsing techniques, such as LR parsing [3]. Although FGs generate obviously a subclass of deterministic CF languages and therefore can be parsed by any deterministic pushdown machine, typically a shift-reduce one [3], we are not aware of a family of automata that perfectly matches the generative power of this class of grammars. On the other hand, operator precedence parsers are still used today, thanks to their elegant simplicity and efficiency. For instance, they are present in Parrot, Perl 6's virtual machine, as part of the Parser Grammar Engine (PGE); in GCC's C and C++ hand-coded parsers, for managing arithmetic expressions.[1]

Quite recently we realized strong relations between these two seemingly unrelated families of languages; precisely we showed that: VPLs are a proper subclass of languages defined by FGs (i.e. Floyd Languages, or FLs in short), and coincide with those languages that can be generated by FGs characterized by a well precise shape of operator precedence matrix (OPM). The inclusion relation is effective in that a FG can be algorithmically derived form a VPA and conversely a VPA can be obtained by a FG whose OPM satisfies the restriction [4].

FLs enjoy all typical closure properties of regular languages that motivated the study of VPLs and other related families [5,6,7]. Precisely, closure w.r.t. Boolean operations was proved a long time ago in [8], whereas closure under concatenation, Kleene star, and other typical algebraic operations has been investigated only recently under the novel interest ignited by the above remark [9]. Thus, the old-fashioned FLs turned out to be the largest known class of deterministic context-free languages that enjoy closure under all traditional language operations. Another reason why, in our opinion, FLs are far from obsolete and uninteresting in these days is that, unlike most other deterministic languages of practical use, they can be parsed not necessarily left-to-right, thus offering interesting opportunities, e.g., to exploit parallelism and incrementality [3].

In this paper we provide another missing tile of the "old and new puzzle", namely we introduce a novel class of stack-based automata perfectly carved on the generation mechanism of FGs, which too we name in honor of Robert Floyd. Not surprisingly they inherit some features of VPAs (mainly a clear separation between push and pop operations) and maintain some typical behavior of shift-reduce parsing algorithms; however, they also exhibit some distinguishing features and imply some non-trivial technicalities to derive them automatically from FGs and conversely.

The availability of a precise family of automata allows to apply to FLs the now familiar ω-extension –a further extension of Kleene $*$ operation–, i.e., the definition of languages of infinite strings and the various criteria for their acceptance or rejection by recognizing devices. ω-languages are now more and more important to deal with never-ending computations such as operating systems, web-services, embedded applications, etc. Thus, we also introduce the ω-version of FLs and we show their potential in terms of modeling the behavior of some realistic systems.

The paper is structured as follows: Section 2 recalls basic definitions on Floyd's grammars; Section 3 introduces Floyd automata (FAs) and shows that, as well as FSMs and VPAs, but unlike pushdown automata, their deterministic version is not less powerful than the nondeterministic counterpart; Section 4 provides effective constructions to derive a FA from a FG and conversely; Section 5 extends the definition of FLs to sets of infinite strings by applying to FAs the well-known concepts of ω-behavior and acceptance; finally Section 6 draws some conclusions.

[1] The interested reader may find more information at `http://gcc.gnu.org`, and `http://www.parrot.org`, respectively.

2 Preliminaries

Let Σ be an alphabet. The empty string is denoted ε. A *context-free* (CF) grammar is a 4-tuple $G = (N, \Sigma, P, S)$, where N is the nonterminal alphabet, P the rule (or production) set, and S the axiom. An *empty rule* has ε as the right hand side (r.h.s.). A *renaming rule* has one nonterminal as r.h.s. A grammar is *reduced* if every rule can be used to generate some string in Σ^*. It is *invertible* if no two rules have identical r.h.s.

The following naming convention will be adopted, unless otherwise specified: lowercase Latin letters $a, b, \ldots$ denote terminal characters; uppercase Latin letters $A, B, \ldots$ denote nonterminal characters; letters $u, v, \ldots$ denote terminal strings; and Greek letters $\alpha, \ldots, \omega$ denote strings over $\Sigma \cup N$. The strings may be empty, unless stated otherwise.

A rule is in *operator form* if its r.h.s has no adjacent nonterminals; an *operator grammar* (OG) contains just such rules. Any CF grammar admits an equivalent OG, which can be also assumed to be invertible [10,11].

The coming definitions for operator precedence grammars [1], here renamed *Floyd Grammars* (FG), are from [8]. We refer the reader unfamiliar with precedence grammars and parsing techniques to [3], that contains an easily readable, practical description of FGs.

For an OG G and a nonterminal A, the *left and right terminal sets* are

$$\mathcal{L}_G(A) = \{a \in \Sigma \mid A \overset{*}{\Rightarrow} Ba\alpha\} \qquad \mathcal{R}_G(A) = \{a \in \Sigma \mid A \overset{*}{\Rightarrow} \alpha aB\}$$

where $B \in N \cup \{\varepsilon\}$ and $\Rightarrow$ denotes the derivation relation. The grammar name G will be omitted unless necessary to prevent confusion.

R. Floyd took inspiration from the traditional notion of precedence between arithmetic operators in order to define a broad class of languages, such that the shape of the derivation tree is solely determined by a binary relation between terminals that are consecutive, or become consecutive after a bottom-up reduction step.

For an OG G, let α, β range over $(N \cup \Sigma)^*$ and $a, b \in \Sigma$. Three binary operator precedence (OP) relations are defined:

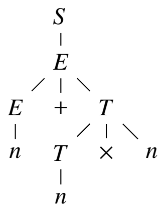
$$\begin{aligned} \text{equal in precedence: } & a \doteq b \iff \exists A \to \alpha aBb\beta, B \in N \cup \{\varepsilon\} \\ \text{takes precedence: } & a \gtrdot b \iff \exists A \to \alpha Db\beta, D \in N \text{ and } a \in \mathcal{R}_G(D) \\ \text{yields precedence: } & a \lessdot b \iff \exists A \to \alpha aD\beta, D \in N \text{ and } b \in \mathcal{L}_G(D) \end{aligned}$$

For an OG G, the *operator precedence matrix* (OPM) $M = OPM(G)$ is a $|\Sigma| \times |\Sigma|$ array that with each ordered pair (a, b) associates the set M_{ab} of OP relations holding between a and b.

Definition 1. *G is an* operator precedence *or* Floyd grammar *(FG) if, and only if, $M = OPM(G)$ is a conflict-free matrix, i.e., $\forall a, b$, $|M_{ab}| \leq 1$.*

Example 1. Arithmetic expressions with prioritized operators, a classical construct, are presented in a simple variant without parentheses. Figure 1 presents the productions of the grammar (left) and the derivation tree of expression $n + n \times n$ (center). We see that

$$\begin{array}{l} S \to E \\ E \to E + T \mid T \times n \mid n \\ T \to T \times n \mid n \end{array}$$



$$\begin{array}{c|ccc} & n & + & \times \\ \hline n & & \gtrdot & \gtrdot \\ + & \lessdot & \gtrdot & \lessdot \\ \times & \doteq & & \end{array}$$

Fig. 1. The Floyd grammar for arithmetic expressions without parentheses

$\times \doteq n$ because they appear in the right-hand side of the same production. Analogously, $+ \lessdot n$ since + is sibling of a node with label T and $n \in \mathcal{L}_G(T)$. The complete OPM is shown in Figure 1 (right).

The equal in precedence relations of a FG alphabet are connected with an important parameter of the grammar, namely the length of the right hand sides of the rules. Clearly, a rule $A \to A_1 a_1 \ldots A_t a_t A_{t+1}$, where each A_i is a possibly missing nonterminal, is associated with relations $a_1 \doteq a_2 \doteq \ldots \doteq a_t$. If the $\doteq$ relation is cyclic, there is no finite bound on the length of the r.h.s of a production. Otherwise the length is bounded by $2 \cdot c + 1$, where $c \geq 1$ is the length of the longest $\doteq$-chain. In this paper, for the sake of simplicity and brevity we assume that all precedence matrices are $\doteq$-cycle free. In the case of FGs this prevents the risk of r.h.s of unbounded length [8], in the case of FAs we will see that it avoids a priori the risk of an unbounded sequence of push operations onto the stack matched by only one pop operation. The hypothesis of $\doteq$-cycle freedom could be replaced by weaker ones, such as a bound on r.h.s, as it happens with FGs, at the price of heavier notation, constructions, and proofs.

Definition 2. *A FG is in Fischer normal form [12] if it is invertible, the axiom S does not occur in the r.h.s. of any rule, no empty rule exists except possibly $S \to \varepsilon$, the other rules having S as l.h.s are renaming, and no other renaming rules exist.*

OPMs play a fundamental role in deterministic parsing of FGs. Thus in the view of defining automata to parse FLs we pair them with the alphabet somewhat mimicking VPL's approach where the terminal alphabet is partitioned into calls, returns, and internals [13]. To this goal, we use a special symbol # not in Σ to mark the beginning and the end of any string. This is consistent with the typical operator parsing technique that requires the lookback and lookahead of one character to determine the precedence relation [3]. The precedence relation in the OPM are extended to include # in the normal way.

Definition 3. *An* operator precedence alphabet *is a pair (Σ, M) where Σ is an alphabet and M is a conflict-free operator precedence matrix, i.e. a $|\Sigma \cup \{\#\}|^2$ array that with each ordered pair (a, b) associates at most one of the operator precedence relations: $\doteq$, $\lessdot$ or $\gtrdot$.*

For $u, v \in \Sigma^*$ we write $u \lessdot v$ if $u = xa$ and $v = by$ with $a \lessdot b$. Similarly for the other precedence relations.

3 Floyd Automata

Definition 4. *A nondeterministic* precedence automaton *(or Floyd automaton) is given by a tuple:* $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$ *where:*

- (Σ, M) *is a precedence alphabet,*
- Q *is a set of states (disjoint from* Σ*),*
- $I \subseteq Q$ *is a set of initial states,*
- $F \subseteq Q$ *is a set of final states,*
- $\delta : Q \times (\Sigma \cup Q) \to 2^Q$ *is the transition function.*

The transition function can be seen as the union of two disjoint functions:

$$\delta_{\text{push}} : Q \times \Sigma \to 2^Q \qquad \delta_{\text{flush}} : Q \times Q \to 2^Q$$

A nondeterministic precedence automaton can be represented by a graph with Q as the set of vertices and $\Sigma \cup Q$ as the set of edge labellings: there is an edge from state q to state p labelled by $a \in \Sigma$ if and only if $p \in \delta_{push}(q, a)$ and there is an edge from state q to state p labelled by $r \in Q$ if and only if $p \in \delta_{flush}(q, r)$. To distinguish flush transitions from push transitions we denote the former ones by a double arrow.

To define the semantics of the automaton, we introduce some notations. We use letters $p, q, p_i, q_i, \ldots$ for states in Q and we set $\Sigma' = \{a' \mid a \in \Sigma\}$; symbols in Σ' are called *marked* symbols. Let $\Gamma = (\Sigma \cup \Sigma' \cup \{\#\}) \times Q$; we denote symbols in Γ as $[a\ q]$, $[a'\ q]$, or $[\#\ q]$, respectively. We set $symbol([a\ q]) = symbol([a'\ q]) = a$, $symbol([\#\ q]) = \#$, and $state([a\ q]) = state([a'\ q]) = state([\#\ q]) = q$. Given a string $\beta = B_1 B_2 \ldots B_n$ with $B_i \in \Gamma$, we set $state(\beta) = state(B_n)$.

We call a *configuration* any pair $C = \langle \beta\ ,\ w \rangle$, where $\beta = B_1 B_2 \ldots B_n \in \Gamma^*$, $symbol(B_1) = \#$, and $w = a_1 a_2 \ldots a_m \in \Sigma^* \#$. A configuration represents both the contents β of the stack and the part of input w still to process. We also set $top(C) = symbol(B_n)$ and $input(C) = a_1$.

A computation of the automaton is a finite sequence of moves $C \vdash C_1$; there are three kinds of moves, depending on the precedence relation between $top(C)$ and $input(C)$:

push move:
if $top(C) \doteq input(C)$ then $\langle \beta\ ,\ aw \rangle \vdash \langle \beta[a\ q]\ ,\ w \rangle$, $\forall q \in \delta_{push}(state(\beta), a)$;

mark move:
if $top(C) \lessdot input(C)$ then $\langle \beta\ ,\ aw \rangle \vdash \langle \beta[a'\ q]\ ,\ w \rangle$, $\forall q \in \delta_{push}(state(\beta), a)$;

flush move:
if $top(C) \gtrdot input(C)$ then let $\beta = B_1 B_2 \ldots B_n$ with $B_j = [x_j\ q_j]$, $x_j \in \Sigma \cup \Sigma'$ and let i the greatest index such that B_i belongs to $\Sigma' \times Q$. Then

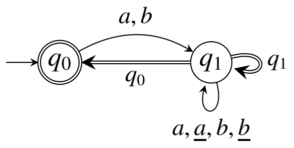
$$\langle \beta\ ,\ aw \rangle \vdash \langle B_1 B_2 \ldots B_{i-2}[x_{i-1}\ \ ],\ aw \rangle,\ \forall q \in \delta_{flush}(q_n, q_{i-1}).$$

Push and mark moves both push the input symbol on the top of the stack, together with the new state computed by δ_{push}; such moves differ only in the marking of the symbol on top of the stack. The flush move is more complex: the symbols on the top of the stack are removed until the first marked symbol (*included*), and the state of the next symbol below them in the stack is updated by δ_{flush} according to the pair of states that delimit the portion of the stack to be removed; notice that in this move the input symbol is not relevant and it remains available for the following move.

Finally, we say that a configuration $[\# \, q_I]$ is *starting* if $q_I \in I$ and a configuration $[\# \, q_F]$ is *accepting* if $q_F \in F$. The language accepted by the automaton is defined as:

$$L(\mathcal{A}) = \left\{ x \mid \langle [\# \, q_I] \, , \, x\# \rangle \overset{*}{\vdash} \langle [\# \, q_F] \, , \, \# \rangle, q_I \in I, q_F \in F \right\}.$$

Example 2. The automaton depicted in Figure 2 accepts the Dyck language L_D of balanced strings of parentheses, with two parentheses pairs $a, \underline{a}$, and $b, \underline{b}$. The same figure also shows an accepting computation on input $aba\underline{aba}a\underline{a}$.



$$\begin{array}{c|ccccc}
 & a & \underline{a} & b & \underline{b} & \# \\
\hline
a & \lessdot & \doteq & \lessdot & & \\
\underline{a} & \lessdot & \gtrdot & \lessdot & \gtrdot & \gtrdot \\
b & \lessdot & & \lessdot & \doteq & \\
\underline{b} & \lessdot & \gtrdot & \lessdot & \gtrdot & \gtrdot \\
\# & \lessdot & & \lessdot & & \doteq
\end{array}$$

	$\langle [\# \, q_0]$	$, \; aba\underline{aba}a\underline{a}\#\rangle$
mark	$\langle [\# \, q_0][a' \; q_1]$	$, \; ba\underline{aba}a\underline{a}\#\rangle$
mark	$\langle [\# \, q_0][a' \; q_1][b' \; q_1]$	$, \; a\underline{aba}a\underline{a}\#\rangle$
mark	$\langle [\# \, q_0][a' \; q_1][b' \; q_1][a' \; q_1]$	$, \; \underline{aba}a\underline{a}\#\rangle$
push	$\langle [\# \, q_0][a' \; q_1][b' \; q_1][a' \; q_1][\underline{a} \, q_1]$	$, \; \underline{ba}a\underline{a}\#\rangle$
flush	$\langle [\# \, q_0][a' \; q_1][b' \; q_1]$	$, \; \underline{ba}a\underline{a}\#\rangle$
push	$\langle [\# \, q_0][a' \; q_1][b' \; q_1][\underline{b} \, q_1]$	$, \; \underline{a}a\underline{a}\#\rangle$
flush	$\langle [\# \, q_0][a' \; q_1]$	$, \; \underline{a}a\underline{a}\#\rangle$
push	$\langle [\# \, q_0][a' \; q_1][\underline{a} \, q_1]$	$, \; a\underline{a}\#\rangle$
mark	$\langle [\# \, q_0][a' \; q_1][\underline{a} \, q_1][a' \; q_1]$	$, \; \underline{a}\#\rangle$
push	$\langle [\# \, q_0][a' \; q_1][\underline{a} \, q_1][a' \; q_1][\underline{a} \, q_1]$	$, \; \#\rangle$
flush	$\langle [\# \, q_0][a' \; q_1][\underline{a} \, q_1]$	$, \; \#\rangle$
flush	$\langle [\# \, q_0]$	$, \; \#\rangle$

Fig. 2. Automaton, precedence matrix, and example of computation for language L_D

A Floyd automaton is called *deterministic* when $\delta_{\text{push}}(q, a)$ and $\delta_{\text{flush}}(q, p)$ have at most one element, for every $q, p \in Q$ and $a \in \Sigma$. Here we prove that deterministic Floyd automata are equivalent to nondeterministic ones, with a power-set construction similar to the one used for classical finite state automata.

Theorem 1. *Deterministic Floyd automata are equivalent to nondeterministic ones.*

Given a nondeterministic automaton $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$, consider the deterministic automaton $\tilde{\mathcal{A}} = \langle \Sigma, M, \tilde{Q}, \tilde{I}, \tilde{F}, \tilde{\delta} \rangle$ where:

- $\tilde{Q} = 2^Q$ is the set of subsets of Q,
- $\tilde{I} = I \in \tilde{Q}$ is the set of initial states of $\mathcal{A}$,
- $\tilde{F} = \{ J \subseteq Q \mid J \cap F \neq \emptyset \} \subseteq \tilde{Q}$, i.e. $\tilde{F}$ is the set of subsets of Q containing at least one final state of $\mathcal{A}$,
- $\tilde{\delta} : \tilde{Q} \times (\Sigma \cup \tilde{Q}) \to \tilde{Q}$ is the transition function defined as follows. The push transition $\tilde{\delta}_{\text{push}} : \tilde{Q} \times \Sigma \to \tilde{Q}$ is defined by

$$\tilde{\delta}_{\text{push}}(J, a) = \bigcup_{p \in J} \delta(p, a);$$

whereas the flush transition $\tilde{\delta}_{\text{flush}} : \tilde{Q} \times \tilde{Q} \to \tilde{Q}$ is defined only on pairs $(J, K) \in \tilde{Q} \times \tilde{Q}$ such that $\delta_{\text{flush}}(p, q_1) = \delta_{\text{flush}}(p, q_2)$ for every $p \in J$ and $q_1, q_2 \in K$; in this case we set

$$\tilde{\delta}_{\text{flush}}(J, K) = \bigcup_{p \in J} \delta_{\text{flush}}(p, q)$$

where q is any element of K.

Theorem 1 is a direct consequence of the following two lemmata:

Lemma 1. *For every path $q_0 \xrightarrow{r_0} q_1 \xrightarrow{r_1} q_2 \xrightarrow{r_2} \ldots \xrightarrow{r_n} q_n$ in $\mathcal{A}$, where $q_i \in Q$ and $r_i \in \Sigma \cup Q$, there is a path $J_0 \xrightarrow{s_0} J_1 \xrightarrow{s_1} J_2 \xrightarrow{s_2} \ldots \xrightarrow{s_n} J_n$ in $\tilde{\mathcal{A}}$, where $J_i \ni q_i$, and $s_i = r_i$ if $r_i \in \Sigma$ or $s_i = \{r_i\}$ if $r_i \in Q$.*

Proof. It is enough to set $J_0 = \{q_0\}$ and $J_i = \delta(q_{i-1}, s_i)$ for every $i \geq 1$. Notice that, in this definition, $\delta = \delta_{\text{push}}$ if $s_i \in \Sigma$, while $\delta = \delta_{\text{flush}}$ if $s_i \in Q$.

Lemma 2. *For every path $J_0 \xrightarrow{s_0} J_1 \xrightarrow{s_1} J_2 \xrightarrow{s_2} \ldots \xrightarrow{s_n} J_n$ in $\tilde{\mathcal{A}}$, with $s_i \in \Sigma \cup \tilde{Q}$, and for every $q_n \in J_n$, there is a path $q_0 \xrightarrow{r_0} q_1 \xrightarrow{r_1} q_2 \xrightarrow{r_2} \ldots \xrightarrow{r_n} q_n$ in $\mathcal{A}$, where $q_i \in J_i$, and $r_i = s_i$ if $s_i \in \Sigma$ or $r_i \in s_i$ if $s_i \in \tilde{Q}$.*

Proof. We reason by induction on the length n of the path. If $n = 1$, let $J_0 \xrightarrow{s_0} J_1$ and $q_1 \in J_1$. First consider the case $s_0 \in \Sigma$. Then $J_1 = \tilde{\delta}_{\text{push}}(J_0, s_0) = \cup_{p \in J_0} \delta_{\text{push}}(p, s_0)$. Since $q_1 \in J_1$, there exists $q_0 \in J_0$ such that $q_1 \in \delta_{\text{push}}(q_0, s_0)$ and hence $q_0 \xrightarrow{s_0} q_1$ in $\mathcal{A}$. In the case $x \in \tilde{Q}$, we have $J_1 = \tilde{\delta}_{\text{flush}}(J_0, s_0) = \bigcup_{p \in J_0} \delta_{\text{flush}}(p, r_0)$ for any $r_0 \in s_0$; then, since $q_1 \in J_1$, there exists $q_0 \in J_0$ such that $q_1 \in \delta_{\text{flush}}(q_0, r_0)$ and hence $q_0 \xrightarrow{r_0} q_1$ in $\mathcal{A}$.

If $n \geq 1$, consider $J_0 \xrightarrow{s_0} J_1 \xrightarrow{s_1} J_2 \xrightarrow{s_2} \ldots \xrightarrow{s_n} J_n$ in $\tilde{\mathcal{A}}$, $q_n \in J_n$, and assume by induction that in $\mathcal{A}$ there is a path $q_1 \xrightarrow{r_1} q_2 \xrightarrow{r_2} \ldots \xrightarrow{r_n} q_n$ with $q_i \in J_i$, and $r_i = s_i$ if $s_i \in \Sigma$ or $r_i \in s_i$ if $s_i \in \tilde{Q}$. In the case $s_0 \in \Sigma$ we have $q_1 \in J_1 = \tilde{\delta}_{\text{push}}(J_0, s_0) = \bigcup_{p \in J_0} \delta_{\text{push}}(p, s_0)$, hence there exists $q_0 \in J_0$ such that $q_1 \in \delta_{\text{push}}(q_0, s_0)$; thus $q_0 \xrightarrow{s_0} q_1$ in $\mathcal{A}$. In the case $s_0 \in \tilde{Q}$ we have $q_1 \in J_1 = \tilde{\delta}_{\text{flush}}(J_0, s_0) = \bigcup_{p \in J_0} \delta_{\text{flush}}(p, r_0)$ for any $r_0 \in s_0$; then there exists $q_0 \in J_0$ such that $q_1 \in \delta_{\text{flush}}(q_0, r_0)$ and hence $q_0 \xrightarrow{r_0} q_1$ in $\mathcal{A}$.

At this point the theorem follows immediately by considering paths beginning in a initial state and ending in a final state, and observing that the stacks of the two automata evolve in parallel.

4 Floyd Automata vs. Floyd Grammars

The main result of this paper is the perfect match between FGs and FAs.

4.1 From Floyd Grammars to Floyd Automata

Theorem 2. *Any L generated by a Floyd grammar can be recognized by a Floyd automaton*

We provide a constructive proof of the theorem: given a Floyd grammar G we build an equivalent nondeterministic Floyd automaton $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta \rangle$, whose precedence matrix M is the same as the one associated with G. A successful computation of $\mathcal{A}$ will correspond to a derivation tree in G: intuitively, a push transition tries to guess the parent of the symbol currently under the input head (i.e. it determines the l.h.s of a rule of G whose r.h.s contains the current symbol); a flush transition is performed whenever the r.h.s of a rule is completed, and determines the corresponding l.h.s., thus confirming some previous guesses.

In order to keep the construction as simple as possible, we avoid introducing any optimization. Also, without loss of generality, we assume that the grammar $G = \langle \Sigma, N, P, S \rangle$ satisfies the following properties: the axiom S does not occur in the r.h.s. of any rule, no empty rule exists except possibly $S \rightarrow \varepsilon$, the other rules having S as l.h.s are renaming, and no other renaming rules exist (in other words, we assume that the G is in Fischer normal form except it is not necessarily invertible).

First of all, we introduce some notation. Enumerate the productions as follows: for any nonterminal $A \in N$, let $P_1(A)$, $P_2(A)$, $\ldots P_{n(A)}(A)$ be the productions having A as l.h.s. (i.e. $n(A)$ is the number of productions having A as l.h.s.). Then, consider the set of *extended nonterminals* $EN = \{A_i \mid A \in N, i = 1, 2, \ldots n(A)\}$ and define $Q = EN \times (EN \cup \{\perp\})$, where $\perp$ is a new symbol whose meaning is *undefined*. To distinguish between nonterminals and extended nonterminals, we will use capital letters $A, B, C, \ldots$ and $X, Y, Z, \ldots$, respectively.

When considering derivation trees of G, we label internal nodes with extended nonterminals (where the subscript of the nonterminal corresponds to the rule applied in the node). Moreover, with a slight abuse of notation, we sometimes confuse nodes and their labels, using the above convention also for internal nodes and leaves.

To define the push transition function $\delta_{push} : Q \times \Sigma \rightarrow 2^Q$, consider any derivation tree τ of G with any leaf a and let X be a's parent in τ. Figure 3 represents the various configurations that τ may exhibit.

– Case 0: if there is no leaf that precedes a in the in-order visit of τ and has depth not greater than a's depth, then let Y be the topmost ancestor of X, i.e., $Y = S_i$ for some i; this also means that $\# \lessdot a$;
– Otherwise, let b be the rightmost such leaf, and let Y be y's parent. Notice that, G being an operator grammar, Y is the nearest common ancestor of a and b. Then there are two possibilities:
  • Case 1: $X = Y$, i.e. $b \doteq a$;
  • Case 2: $X \neq Y$, and in this case b has lower depth than a, so $b \lessdot a$.

In all cases, node Z may be missing, or there may be other leaves between b and a (namely, Z's descendants); let $\hat{Z} = \perp$ if Z is missing, $\hat{Z} = Z$ otherwise. Then, for each such triple (a, X, Y), define the (a, X, Y)*-push transition*:

$$\delta_{push}((Y, \hat{Z}), a) \ni \begin{cases} (X, X) \text{ if } a \text{ is the rightmost child of } X, \\ (X, \perp) \text{ otherwise.} \end{cases}$$

Hence, a push transition essentially determines the parent of the symbol under the input head (actually, a "candidate" parent, since the automaton is non-deterministic).

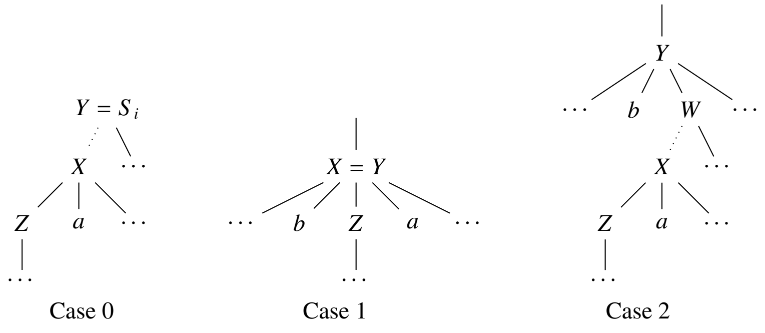


Fig. 3. Derivation tree configurations for the push transition function (nodes labelled as ... could be missing)

A similar construction holds for the flush transition function $\delta_{flush} : Q \times Q \to 2^Q$. For every derivation tree with internal node X, let f and ℓ be the first and last child, respectively, of node X. Notice that both f and ℓ may be either internal nodes or leaves. Then there are two possibilities, as depicted in Figure 4:

– Case 3: there is no leaf at the left of X, then let Y be the topmost ancestor of X, i.e., $Y = S_i$ for some i;
– Case 4: otherwise, let b be the rightmost leaf at the left of X and let Y be b's parent (again, notice that Y is the nearest common ancestor of X and b, G being an operator grammar).

Also, let $\ell_{/X}$ be ℓ if ℓ is an internal node, X otherwise; let $\tilde{f}$ be f if f is an internal node, $\perp$ otherwise. Then, for each such pair (X, Y) define the (X, Y)-*flush transition*:

$$\delta_{flush}((X, \ell_{/X}), (Y, \tilde{f})) \ni (Y, X).$$

Hence, the state computed by a flush transition contains two pieces of information: the first component determines the nearest ancestor of both X and b (or the axioms if b does not exist), while the second component determines the nonterminal corresponding to the r.h.s. just completed.

Finally, initial and final states are defined as follows.

$$I = \{(S_i, \perp) \mid 1 \le i \le n(S)\}, \qquad F = \{(S_i, A_j) \mid S \to A \in P, 1 \le i \le n(S), 1 \le j \le n(A)\}.$$

Notice that the above construction is effective. All triples (a, X, Y) involved by some push transition can be found starting from any rule $X \to \alpha$ with α containing a: if a is not the leftmost terminal of α, then take the triple (a, X, X), else apply backwards any rule with r.h.s starting with X and extend this process until all productions have been examined. Similarly for the flush transitions.

Example 3. Let G be the grammar introduced in Example 1. Following the above construction, number the rules of the grammar in the order they appear in the definition of

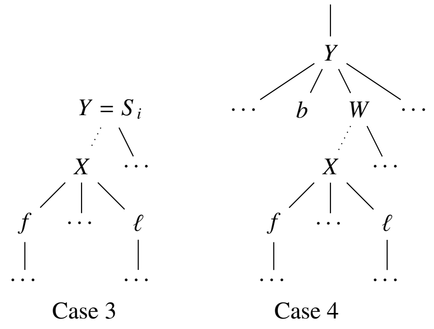


Fig. 4. Derivation tree configurations for the flush transition function (all nodes marked as ... could be missing)

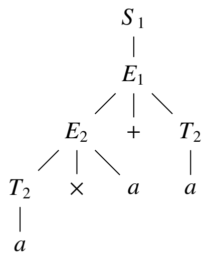
G (for instance, $P_2(E)$ is $E \to T \times a$). The transitions defined by the derivation tree of string $a \times a + a$, depicted in Figure 5 (left), are the following:

$$\begin{array}{ll}
\delta_{push}((S_1,\bot),a) \ni (T_2,T_2) & \\
\delta_{push}((S_1,T_2),\times) \ni (E_2,\bot) & \delta_{flush}((T_2,T_2),(E_1,\bot)) \ni (E_1,T_2) \\
\delta_{push}((S_1,E_2),+) \ni (E_1,\bot) & \delta_{flush}((T_2,T_2),(S_1,T_2)) \ni (S_1,T_2) \\
\delta_{push}((E_2,\bot),a) \ni (E_2,E_2) & \delta_{flush}((E_2,E_2),(S_1,T_2)) \ni (S_1,E_2) \\
\delta_{push}((E_1,\bot),a) \ni (T_2,T_2) & \delta_{flush}((E_1,T_2),(S_1,E_2)) \ni (S_1,E_1)
\end{array}$$

The first one is the (a, T_2, S_1)-push transition obtained by starting from the left-most leaf (Case 0). Case 0 occurs also for the second and third push transitions, obtained considering the leaves labeled by $\times$ and $+$, respectively. The other push transitions represent instances of Cases 1 and 2, in this order. As far as flush transitions are concerned, Case 4 occurs only in the first stated transition, with $X = T_2$, $b = +$ and $Y = E_1$, whereas all other productions represent instances of Case 3. Hence, on input $a \times a + a$, the automaton $\mathcal{A}$ obtained from G may execute the computation represented in Figure 5 (right).

The equivalence between G and the automaton described above is based on the following lemma, whose proof is omitted because of space reasons. As usual we set $\Gamma = (\Sigma \cup \Sigma') \times Q = (\Sigma \cup \Sigma') \times (EN \times (EN \cup \{\bot\}))$ and we denote an element in Γ as $[a\ (X,Y)]$. To avoid an excessively cumbersome notation, when describing the transitions between configurations, we omit the extreme parts (i.e. the lower part of the stack and a suffix of the input string) which are not affected by the computation.

We define the *depth of a computation* $C_1 \overset{*}{\vdash} C_2$ as the maximum number of marked symbols in one of the traversed configurations, minus the number of marked symbol on the stack in configuration C_1; we define the *depth of a derivation* $W \overset{*}{\Rightarrow} \alpha$ as the depth of the corresponding derivation tree. When useful, we make the depth h of a computation or a derivation explicit as in $C_1 \overset{[h]}{\vdash} C_2$ and $X \overset{[h]}{\Rightarrow} \alpha$.



$$\begin{array}{lll}
 & \langle [\#\,(S_1,\bot)] & ,\ a \times a + a\,\#\rangle \\
\text{mark} & \langle [\#\,(S_1,\bot)][a'\,(T_2,T_2)] & ,\ \times a + a\,\#\rangle \\
\text{flush} & \langle [\#\,(S_1,T_2)] & ,\ \times a + a\,\#\rangle \\
\text{mark} & \langle [\#\,(S_1,T_2)][\times'\,(E_2,\bot)] & ,\ a + a\,\#\rangle \\
\text{push} & \langle [\#\,(S_1,T_2)][\times'\,(E_2,\bot)][a\,(E_2,E_2)] & ,\ + a\,\#\rangle \\
\text{flush} & \langle [\#\,(S_1,E_2)] & ,\ + a\,\#\rangle \\
\text{mark} & \langle [\#\,(S_1,E_2)][+'\,(E_1,\bot)] & ,\ a\,\#\rangle \\
\text{mark} & \langle [\#\,(S_1,E_2)][+'\,(E_1,\bot)][a'\,(T_2,T_2)] & ,\ \#\rangle \\
\text{flush} & \langle [\#\,(S_1,E_2)][+'\,(E_1,T_2)] & ,\ \#\rangle \\
\text{flush} & \langle [\#\,(S_1,E_1)] & ,\ \#\rangle
\end{array}$$

Fig. 5. Derivation tree (left) and computation (right) for the string $a \times a + a$

Lemma 3. *Let Y, W be extended nonterminals of G, $v \in \Sigma^*$, $a \lessdot v \gtrdot b$, and $\bar{a} \in \{a, a'\}$. Then for all $h \geq 1$:*

$$\langle [\bar{a}\,(Y,\bot)]\,,\ vb\rangle \overset{[h]}{\vdash} \langle [\bar{a}\,(Y,W)]\,,\ b\rangle \quad \textit{iff} \quad \exists \alpha,\beta \text{ such that } Y \to \alpha a W \beta,\ W \overset{[h]}{\Rightarrow} v \text{ in } G.$$

From the lemma the theorem easily follows by using a special case $S \to A$ (with implicit # as a and b).

4.2 From Floyd Automata to Floyd Grammars

Given a Floyd automaton $\mathcal{A} = \langle \Sigma, M, Q, I, F, \delta\rangle$, we show how to build an equivalent Floyd grammar G having operator precedence matrix M. In order to keep the construction as easy as possible, w.l.o.g we assume that M is $\doteq$-acyclic. Remind that, as discussed in Section 2, this hypothesis could be replaced by weaker ones.

We need some notation and definitions. First of all, we shall represent a push transition with a simple arrow $\rightarrow$, a flush transition with a double arrow $\Rightarrow$, and a path defined by a sequence of transitions with a wavy arrow $\rightsquigarrow$.

We define *chains* in $\mathcal{A}$ recursively. A *simple chain* is a word $a_0a_1a_2\ldots a_na_{n+1}$, written as $\langle^{a_0}a_1a_2\ldots a_n{}^{a_{n+1}}\rangle$, such that: $a_0, a_{n+1} \in \Sigma \cup \{\#\}$, $a_i \in \Sigma$ for every $i = 1, 2, \ldots n$, $M_{a_0,a_{n+1}} \neq \emptyset$, and $a_0 \lessdot a_1 \doteq a_2 \ldots a_{n-1} \doteq a_n \gtrdot a_{n+1}$. A *composed chain* in $\mathcal{A}$ is a word $a_0x_0a_1x_1a_2\ldots a_nx_na_{n+1}$, where $\langle^{a_0}a_1a_2\ldots a_n{}^{a_{n+1}}\rangle$ is a simple chain, and $x_i \in \Sigma^*$ is the empty word or is such that $\langle^{a_i}x_i{}^{a_{i+1}}\rangle$ is a chain (simple or composed), for every $i = 0, 1, \ldots, n-1$. Such a composed chain will be written as $\langle^{a_0}x_0a_1x_1a_2\ldots a_nx_n{}^{a_{n+1}}\rangle$.

We call a *support* for the simple chain $\langle^{a_0}a_1a_2\ldots a_n{}^{a_{n+1}}\rangle$ any path in $\mathcal{A}$ of the form

$$q_0 \xrightarrow{a_1} q_1 \longrightarrow \ldots \longrightarrow q_{n-1} \xrightarrow{a_n} q_n \overset{q_0}{\Longrightarrow} q_{n+1} \tag{1}$$

Notice that the label of the last (and only) flush is exactly q_0, i.e. the first state of the path; this flush is executed because of relation $a_n \gtrdot a_{n+1}$. We call a *support for the composed chain* $\langle^{a_0}x_0a_1x_1a_2\ldots a_nx_n{}^{a_{n+1}}\rangle$ any path in $\mathcal{A}$ of the form

$$q_0 \overset{x_0}{\rightsquigarrow} q'_0 \xrightarrow{a_1} q_1 \overset{x_1}{\rightsquigarrow} q'_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} q_n \overset{x_n}{\rightsquigarrow} q'_n \overset{q'_0}{\Longrightarrow} q_{n+1} \tag{2}$$

where, for every $i = 0, 1, \ldots, n$:

– if $x_i \neq \epsilon$, then $q_i \overset{x_i}{\rightsquigarrow} q'_i$ is a support for the chain $\langle^{a_i} x_i {}^{a_{i+1}}\rangle$, i.e., it can be decomposed as $q_i \overset{x_i}{\rightsquigarrow} q''_i \overset{q_i}{\Longrightarrow} q'_i$.
– if $x_i = \epsilon$, then $q'_i = q_i$.

Notice that the label of the last flush is exactly q'_0.

We are now able to define a Floyd grammar $G = \langle \Sigma, N, S, P\rangle$. Nonterminals are the 4-tuples $(a, q, p, b) \in \Sigma \times Q \times Q \times \Sigma$, written as $\langle^a p, q^b\rangle$, plus the axiom S. Rules are built as follows:

– for every support of type (1) of a simple chain, add the rule

$$\langle^{a_0} q_0, q_{n+1}{}^{a_{n+1}}\rangle \longrightarrow a_1 a_2 \dots a_n \ ;$$

if also $a_0 = a_{n+1} = \#$, q_0 is initial, and q_{n+1} is final, add the rule $S \to \langle^{\#} q_0, q_{n+1}{}^{\#}\rangle$;

– for every support of type (2) of a composed chain, add the rule

$$\langle^{a_0} q_0, q_{n+1}{}^{a_{n+1}}\rangle \longrightarrow N_0 a_1 N_1 a_2 \dots a_n N_n \ ;$$

where, for every $i = 0, 1, \dots, n$, $N_i = \langle^{a_i} q_i, q'_i{}^{a_{i+1}}\rangle$ if $x_i \neq \epsilon$ and $N_i = \epsilon$ otherwise.

Notice that the above construction is effective thanks to the hypothesis of $\doteq$-acyclicity of the OPM. This implies that the length of the r.h.s. is bounded (see Section 2); on the other hand, the cardinality of the nonterminal alphabet is finite. Hence there is only a finite number of possible productions for G and only a limited number of chains to be considered.

5 ω-Languages

Having an operational model that defines Floyd Languages, it is now straightforward to introduce extensions to ω-languages.

For instance, the classical Büchi condition of acceptance can be easily adapted to FAs. Consider an infinite word $x \in \Sigma^\omega$, and an *infinite computation* of the automaton $\mathcal{A}_M = \langle \Sigma, M, Q, I, F, \delta\rangle$ on x, i.e. an ω-sequence of configurations $\mathcal{S} = \langle \beta_0 \ , \ x_0\rangle \langle \beta_1 \ , \ x_1\rangle \dots$, such that $\langle \beta_0 \ , \ x_0\rangle = \langle [\# \ q_I] \ , \ x\rangle$, $q_I \in I$ and $\langle \beta_i \ , \ x_i\rangle \vdash \langle \beta_{i+1} \ , \ x_{i+1}\rangle$. We say that $x \in L(\mathcal{A})$ if and only if there exists $q_F \in F$ such that configurations with stack $[\# \ q_F]$ occur infinitely often in $\mathcal{S}$.

Quite naturally, ω-VPLs are a proper subset of this class of languages, as it is shown by the following example.

Example 4. We define here the stack management of a simple programming language that is able to handle nested exceptions. For simplicity, there are only two procedures, called a and b. Calls and returns are denoted by $call_a$, $call_b$, ret_a, ret_b, respectively. During execution, it is possible to install an exception handler *hnd*. The last signal that we use is *rst*, that is issued when an exception occur, or after a correct execution to uninstall the handler. With a *rst* the stack is "flushed", restoring the state right before the last *hnd*. The automaton is presented in Figure 6 (notice that it is an extension of the automaton in Figure 2). It is easy to modify this example to model the case of *unnested* exceptions, to fit with other application contexts.

	$call_a$	ret_a	$call_b$	ret_b	hnd	rst
$call_a$	$\lessdot$	$\doteq$	$\lessdot$		$\lessdot$	$\gtrdot$
ret_a	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$		$\gtrdot$
$call_b$	$\lessdot$		$\lessdot$	$\doteq$	$\lessdot$	$\gtrdot$
ret_b	$\lessdot$	$\gtrdot$	$\lessdot$	$\gtrdot$		$\gtrdot$
hnd	$\lessdot$	$\lessdot$	$\lessdot$	$\lessdot$		$\doteq$
rst	$\gtrdot$	$\gtrdot$	$\gtrdot$	$\gtrdot$		
#	$\lessdot$		$\lessdot$		$\lessdot$	

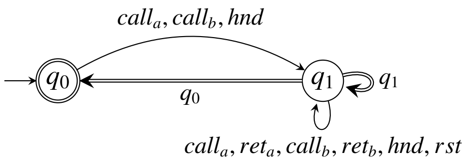


Fig. 6. Precedence matrix and automaton for an ω-language. There is no column indexed by # since words are infinite.

6 Conclusions and Further Research

Recently, we advocated that operator precedence grammars and languages, here renamed after their inventor Robert Floyd, deserve renewed attention in the realm of formal languages. The main reasons to support our claim are:

- The fact that this family of languages properly includes visibly pushdown languages [13], a new family that has been proposed with the main motivation of extending powerful model checking techniques beyond the limits of finite state machines.
- The fact that it enjoys all closure properties with respect to the main algebraic operations that are exhibited by regular languages and VPLs.
- The fact that, unlike other deterministic languages -either strictly more powerful than them, or incomparable with them- such as LR, LL, and simple precedence ones, FLs can be parsed without applying a strictly left-to-right order; this feature becomes particularly relevant in these days since it allows to exploit much better the gains in efficiency offered by massive parallelism.

In this paper we filled a rather surprising "hole" in the theory of these languages, namely the lack of an appropriate family of automata that perfectly matches the generative power of their grammars. We defined FAs with such a goal in mind and we proved their equivalence with FGs. Both facts turned out to be non-trivial jobs and showed further interesting peculiarities of this pioneering family of deterministic languages. A first "byproduct" of the new automata family is the extension of FLs to ω-languages, i.e., languages consisting of infinite strings, a more and more important aspect of formal language theory needed to deal with never ending computations. In this case too FL ω-languages proved to augment the descriptive capabilities of the original VPLs.

As a first step towards applicability of the results presented in this paper, and also to validate our approach with several practical examples, we implemented a simple prototypical tool, called *Flup*. Flup contains an interpreter for non-deterministic Floyd Automata, and a Floyd Grammar to Automata translator, that directly applies the construction presented in Section 4.1. All the examples presented in the paper were tried on, or generated by the tool.[2]

[2] The prototype is freely available at `http://home.dei.polimi.it/pradella`

We are confident that suitable future research will further strengthen the importance of, and motivation for, re-inserting FLs in the main stream of formal language literature. In particular it would be interesting to complete the parallel analysis and comparison with VPLs by investigating a characterization in terms of suitable logic formulas [13]; by this way motivation for, and application of, strong model checking techniques would be further enhanced.

References

1. Floyd, R.W.: Syntactic analysis and operator precedence. Journ. ACM 10(3), 316–333 (1963)
2. Alur, R., Madhusudan, P.: Visibly pushdown languages. In: STOC: ACM Symposium on Theory of Computing, STOC (2004)
3. Grune, D., Jacobs, C.J.: Parsing techniques: a practical guide, p. 664. Springer, New York (2008)
4. Crespi Reghizzi, S., Mandrioli, D.: Algebraic properties of structured context-free languages: old approaches and novel developments. In: WORDS 2009 - 7th Int. Conf. on Words (2009) (preprints), http://arXiv.org/abs/0907.2130
5. Berstel, J., Boasson, L.: Balanced grammars and their languages. In: Brauer, W., Ehrig, H., Karhumäki, J., Salomaa, A. (eds.) Formal and Natural Computing. LNCS, vol. 2300, pp. 3–25. Springer, Heidelberg (2002)
6. Nowotka, D., Srba, J.: Height-deterministic pushdown automata. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 125–134. Springer, Heidelberg (2007)
7. Caucal, D.: Boolean algebras of unambiguous context-free languages. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FSTTCS 2008, Dagstuhl, Germany (2008)
8. Crespi Reghizzi, S., Mandrioli, D., Martin, D.F.: Algebraic properties of operator precedence languages. Information and Control 37, 115–133 (1978)
9. Crespi Reghizzi, S., Mandrioli, D.: Operator precedence and the visibly pushdown property. In: Dediu, A.-H., Fernau, H., Martín-Vide, C. (eds.) LATA 2010. LNCS, vol. 6031, pp. 214–226. Springer, Heidelberg (2010)
10. Harrison, M.A.: Introduction to Formal Language Theory. Addison Wesley, Reading (1978)
11. Salomaa, A.K.: Formal Languages. Academic Press, New York (1973)
12. Fischer, M.J.: Some properties of precedence languages. In: STOC 1969: Proc. First Annual ACM Symp. on Theory of Computing, pp. 181–190. ACM, New York (1969)
13. Alur, R., Madhusudan, P.: Adding nesting structure to words. Journ. ACM 56 (2009)

Orbits of Linear Maps and Regular Languages

Sergey Tarasov and Mikhail Vyalyi

Dorodnitsyn Computing Center of RAS
serge99meister@gmail.com,
vyalyi@gmail.com

Abstract. The chamber hitting problem (CHP) for linear maps consists in checking whether an orbit of a linear map specified by a rational matrix hits a given rational polyhedral set. The CHP generalizes some well-known open computability problems about linear recurrent sequences (e.g., the Skolem problem, the nonnegativity problem). It is recently shown that the CHP is Turing equivalent to checking whether an intersection of a regular language and the special language of permutations of binary words (the permutation filter) is nonempty ($P_{\mathbb{B}}$-realizability problem).

In this paper we present some decidable and undecidable problems closely related to $P_{\mathbb{B}}$-realizability problem thus demonstrating its 'borderline' status with respect to computability.

Keywords: decidability, regular language, linear recurrence.

Introduction

The chamber hitting problem is a straightforward generalization of some well-known open decidability problems concerning integral linear recurrent sequences. It is formulated as follows.

Let Φ be a linear map of a vector space V into itself and let $x \in V$ be a vector in V. The iterations of Φ applied to x define an *orbit* $\mathrm{Orb}_{\Phi}\, x$, i.e. the set

$$\{\Phi^k x : k \in \mathbb{Z}^+\}.$$

Chamber hitting problem (CHP)
INPUT: a square matrix Φ of order d; d-dimensional vector x_0; a family of affine functions $h_1, \ldots, h_m$ on $\mathbb{Q}^d$ and a sign pattern $s \in \{\pm 1, 0\}^m$.
OUTPUT: 'yes' if the orbit $\mathrm{Orb}_{\Phi}\, x_0$ intersects the chamber H_s and 'no' otherwise.

Here *a chamber* is a set $H_s = \{x \in \mathbb{Q}^d : \mathrm{sign}(h_i(x)) = s_i \text{ for } 1 \le i \le m\}$, where $\mathrm{sign}(t)$ is a standard sign function.

Remark 1. We assume that matrices, vectors and affine functions are represented by the component lists, where the components are written in binary.

The CHP is a representative of a wide class of orbit description problems associated with orbits of linear maps. These problems were introduced in recent works [5,8] and are related to problems on linear recurrent sequences. For instance, the CHP is a generalization of the famous Skolem problem.

A linear recurrent sequence (LRS) x_n of a degree d is defined by a recurrence equation $x_n = \sum_{i=1}^{d} a_i x_{n-i}$, when $n > d$, and initial conditions $x_n = b_n$, when $1 \le n \le d$. Here a_i, b_j are constants.

The Skolem problem is perhaps the most known algorithmic problem on linear recurrences. It asks whether integral LRS $\{x_n\}$ contains zero, i.e. whether $x_k = 0$ for some k.

The Skolem problem is proved to be decidable for the degrees ≤ 5 (the cases $d = 3, 4$ are worked out in [6], and the case $d = 5$ is solved in [3]). In the opposite direction, the best hardness result on the Skolem problem is NP-hardness [2].

It is easy to see that the Skolem problem is equivalent to the CHP restricted to one linear function and equality (the chamber H_0).

The main result of [5] states an algorithmic equivalence between the CHP and verifying a particular property of the regular languages. Namely, the property involved consists in checking whether a regular language contains at least one word from a special set of words. We call this set a *permutation filter*. Speaking informally, an arbitrary word from the permutation filter is a concatenation of all binary words of a fixed length n separated by the delimeters.

To be a bit more formal, we define a general problem of *regular realizability*: to check whether a given regular language contains a word of a particular kind. It is convenient to describe the realizability problem as follows. Let L be a language in a finite alphabet Σ. Informally, L encodes a *property* that is checked. The input to the problem of L-realizability is a description of some regular language $R \subseteq \Sigma^*$ and we are asked whether the intersection $L \cap R$ is nonempty. In the sequel we assume that R is given via deterministic automaton accepting it.

The permutation filter $P_{\mathbb{B}}$ is a language over the alphabet $\{\#, 0, 1\}$ of all words $\#w_1\#w_2\#\ldots w_N\#$, where $N = 2^n$, $n \ge 1$, and the set $\{w_1, w_2, \ldots, w_N\}$ consists of all binary words of the length n.

Words from $P_{\mathbb{B}}$ are called *permutation words*. The length n is called the *block rank* of a permutation word.

Theorem 1 ([5, Theorem 2]). CHP *and $P_{\mathbb{B}}$-realizability problem are Turing equivalent.*

The Skolem problem is open for almost eighty years. Using slight abuse of language, presently it falls ‘on the border between decidability and undecidability’ [3]. In this paper we show that analogous ‘borderline’ pattern holds for a more general $P_{\mathbb{B}}$-realizability problem.

Below we present some closely related decidable and undecidable variations of $P_{\mathbb{B}}$-realizability problem. All variations are regular realizability problems. The decidable languages involved consist of binary *block words* separated by the delimiter # (*block languages*). All blocks have the same length (the block rank n). Blocks of a block word form a multiset of binary words of the same length (*block multiset*).

The most natural variations of the permutation filter are *the surjective filter* $S_{\mathbb{B}}$ and *the injective filter* $I_{\mathbb{B}}$.

The former consists of those block words whose block multiset contains all words of the length n.

The latter consists of those block words whose block multiset is a set, i.e. each block appears at most once in a word from $I_{\mathbb{B}}$.

It is clear from the definitions that $I_{\mathbb{B}} \cap S_{\mathbb{B}} = P_{\mathbb{B}}$.

The problems of $I_{\mathbb{B}}$-realizability and $S_{\mathbb{B}}$-realizability are decidable (see Section 2, Theorems 2, 3).

To present an undecidable variation of $P_{\mathbb{B}}$-realizability problem we introduce *a track product of block languages*.

Here we assume that blocks are words over a finite alphabet Σ and a block word consists of blocks of the same length separated by the delimiter #.

Let Σ_1, Σ_2 be finite alphabets. For the alphabet $\Sigma_1 \times \Sigma_2$ there are two natural projections from $(\Sigma_1 \times \Sigma_2)^*$, one to Σ_1^* and the other to Σ_2^*:

$$\begin{aligned} \pi_1 \colon (a_1, b_1)(a_2, b_2) \ldots (a_n, b_n) &\mapsto a_1 a_2 \ldots a_n, \\ \pi_2 \colon (a_1, b_1)(a_2, b_2) \ldots (a_n, b_n) &\mapsto b_1 b_2 \ldots b_n. \end{aligned} \tag{1}$$

The track product of two block languages L_1 (over an alphabet Σ_1) and L_2 (over the alphabet Σ_2) is a block language $L = L_1 \| L_2$ consisting of all block words over the alphabet $\{\#\} \cup \Sigma_1 \times \Sigma_2$ such that the projection π_1 is in the language L_1 and the projection π_2 is in the language L_2. (For consistency we assume that $\pi_1(\#) = \pi_2(\#) = \#$.)

Denote by Per_Σ the block language consisting of all periodic words over the alphabet Σ (all blocks of a word in Per_Σ are equal). And denote by P_Σ the block language consisting of permutation block words over the alphabet Σ (blocks of a word in P_Σ form the set of all words in Σ^n, where n is the block rank).

The Per_Σ-realizability problem is decidable. Actually it is PSPACE-complete as shown in [7]. It turns out that the track product of the Per_{Σ_1} with the P_{Σ_2} is undecidable for $|\Sigma_1| = 2$ and $|\Sigma_2| = 648$ (see Section 3, Theorem 4).

1 Preliminaries

In what follows we need technical constructions from [5]. So we reproduce proofs of two results.

Let $\Gamma(V, E)$ be a digraph. We assume that the edges of the digraph are colored in s colors from the set $\{1, 2, \ldots, s\}$. For a walk τ we define a *weight* $w(\tau) \in \mathbb{Z}^s$ as an s-dimensional integer vector $w(\tau) = (c_1(\tau), c_2(\tau), \ldots, c_i(\tau), \ldots c_s(\tau))$, where $c_i(\tau)$ equals the number of edges colored in the color i along the walk τ.

Walk Weight Hitting Problem (WWHP)
INPUT: a digraph Γ, an s-coloring of the edges of the Γ, two vertices a, b of Γ, an integer matrix Φ of order $s \times s$ and an integer vector x_0 of dimension s.
OUTPUT: 'yes' if the orbit $\mathrm{Orb}_\Phi\, x_0$ intersects the set of weights of all walks from the vertex a to the vertex b and 'no' otherwise.

Lemma 1 ([5, Lemma 5]). *The $P_{\mathbb{B}}$-realizability problem is Turing reducible to the* WWHP *problem.*

Let R be a regular language over the alphabet $\{0, 1, \#\}$ and let A be a deterministic automaton with the state set Q accepting the language R. Let pass to the transition monoid of A. The operation of the automaton is determined by the three maps f_0, f_1, $f_\#$ of the set Q into itself induced by reading respective symbols. Our goal is to check whether $R \cap P_{\mathbb{B}} \neq \varnothing$. Let's express this condition in terms of the maps f_0, f_1, $f_\#$. Let's define $f(w)$, where $w = w_1 w_2 \dots w_\ell \in \{0, 1\}$, as $\prod_{i=1}^{\ell} f_{w_{\ell-i+1}}$.

We denote the initial state of the automaton by q_s and the accepting set by Q_a. The reduction algorithm checks for each accepting state $q_f \in Q_a$ the condition whether it can be reached from the initial state q_s after reading some word from the permutation filter. Formally this condition means that for some word $\# w_1 \# w_2 \# \dots w_N \# \in P_{\mathbb{B}}$ we have

$$q_f = \left(\prod_{i=0}^{N-1} f_\# f(w_{N-i}) \right) f_\# q_s. \tag{2}$$

It is important that to verify the condition (2) we do not need to know the sequence w_i. It is sufficient to compute for each map $g \in Q^Q$ the number $\nu_n(g)$ of its representations in the form $f(w)$, where $w \in \{0, 1\}^n$ (recall that $N = 2^n$). Then the condition (2) is rewritten as

$$q_f = \left(\prod_{i=0}^{N-1} f_\# g_i \right) f_\# q_s, \tag{3}$$

where each map $g \in Q^Q$ occurs exactly $\nu_n(g)$ times in the sequence g_i.

Now we are going to describe the condition (3) in terms of walk weights for a suitable graph. For this purpose we will use the Cayley graphs for monoids.

Let $G = \{g_1, \dots, g_m\} \subseteq Q^Q$ be a set of maps. It generates a monoid M (a monoid operation is a map composition, recall that by definition a monoid contains also the identity map.). By definition the vertices of *the Cayley graph* Γ_G of the monoid are elements of the monoid M and (directed) edges have the form $(h, g_i h)$ for $h \in M$, $g_i \in G$. Note that the edges of the Cayley graph are naturally colored by elements of G. The Cayley graph of a monoid may contain parallel edges as the equality $g_i h = g_j h$ for $i \neq j$ may hold.

Let M_{01} be a semigroup generated by the maps f_0, f_1 of the automaton A. We define the monoid M generated by maps $f_\# f$, $f \in M_{01}$.

Denote by Γ_M the Cayley graph of the monoid M w.r.t. the set of generators $\{f_\# f,\ f \in M_{01}\}$.

It follows from the construction that (3) holds iff there exist an integer n and a walk τ on the digraph Γ_M from the vertex id to a vertex h such that $h(f_\#(q_s)) = q_f$ and $(w(\tau))_{f_\# g} = \nu_n(g)$ holds for all g. (Here id is the identity map.).

It turns out that the integers $\nu_n(g)$ can be expressed as the coordinates of the vectors taken from the orbit of a linear map.

In $\mathbb{Q}$-vector space $\mathbb{Q}(Q^Q)$ equipped with the basis $\{e(f)\}$ indexed by maps $f\colon Q \to Q$ we define a linear map by the action on the basis vectors

$$\varPhi e(f) = e(ff_0) + e(ff_1). \tag{4}$$

(Recall that map composition is taken from the right to the left.).

It is easy to prove by induction on n that $\nu_n(g)$ equals the $e(g)$-coordinate of the vector $\varPhi^n e(\mathrm{id})$ w.r.t. the basis $\{e(f)\}$. Indeed, the case of $n = 0$ is trivial and assuming the claim for $n-1$ we have

$$\begin{aligned}\varPhi^n e(\mathrm{id}) &= \varPhi \sum_{g\in Q^Q} \nu_{n-1}(g)e(g) = \sum_{w\in\{0,1\}^{n-1}} \varPhi e(f(w)) = \\ &= \sum_{w\in\{0,1\}^{n-1}} (e(f(w)f_0) + e(f(w)f_1) = \sum_{w\in\{0,1\}^{n-1}} (e(f(w0)) + e(f(w1)) = \\ &= \sum_{w\in\{0,1\}^{n}} e(f(w)) = \sum_{g\in Q^Q} \nu_n(g)e(g). \end{aligned} \tag{5}$$

Now we conclude that the condition (3) is equivalent to the condition

$$w(\tau) = \varPhi^n x_0 \tag{6}$$

for some n and a walk τ on the digraph $\varGamma_M$ from the vertex id to some vertex h such that $h(f_\#(q_s)) = q_f$.

Remark 2. Note that the size of the monoid M can be less than $|Q|^{|Q|}$ and $\varPhi$ acts on $\mathbb{Q}(Q^Q)$. To fix a difference in vector dimensions we extend the coordinates of walk weights by zero values for $e(g)$ such that $g \notin M$.

Proof (of Lemma 1). The reduction algorithm solves with help of a WWHP-oracle several instances of the WWHP indexed by the accepting states q_f and maps h such that $h(f_\#(q_s)) = q_f$.

An instance has the following input data: a digraph is the Cayley graph $\varGamma_M$, colors are generators, the initial vertex is the identity map id, the terminal vertex is h, the matrix is defined by (4) and the initial vector is $x_0 = e(\mathrm{id})$.

If the answer is positive for some instance from the described set then the answer in the instance of the $P_{\mathbb{B}}$-realizability problem involved is also positive due to (5) and the condition (6).

Otherwise, it follows from the definition of the graph $\varGamma_M$ and (5) that the answer in the instance of the $P_{\mathbb{B}}$-realizability problem is negative. □

An integer cone $\mathbb{N}(v_1,\ldots,v_r)$ is the set of vectors $\sum_{i=1}^r a_i v_i$, where $v_i \in \mathbb{Z}^d$, $a_i \in \mathbb{N}$. (We denote the set of nonnegative integers by $\mathbb{N}$.)

Integer cone Hitting Problem (IHP)
INPUT: a square matrix $\varPhi$ of order d; a d-dimensional vector x_0; a family of vectors $v_i \in \mathbb{Z}^d$, $i = 0, 1, \ldots, r$.
OUTPUT: 'yes' if the orbit $\mathrm{Orb}_\varPhi\, x_0$ intersects the translate of the integer cone $v_0 + \mathbb{N}(v_1,\ldots,v_r)$ and 'no' otherwise.

Lemma 2 ([5, Lemma 6]). *The* WWHP *is Turing reducible to the* IHP*.*

At first we prove the following lemma.

Lemma 3. *Let a, b be vertices of a digraph Γ colored in s colors and let $W(a, b)$ be the set of weights of all walks from the vertex a to the vertex b. The set $W(a, b)$ is a finite union of translates of integer cones in $\mathbb{Z}^s$.*

The list of the cones and the vectors of translation is constructed algorithmically.

Proof. Writing a sequence of colors along a walk gives a word in the s-letter alphabet. The collection of such words for all walks from the vertex a to the vertex b forms a regular language.

The weight of a walk is just the Parikh image of the corresponding word. So the statement of the lemma follows from Parikh's theorem [4]. □

Lemma 3 implies that to check the condition (6) it is sufficient to check several conditions of the form

$$\Phi^n x_0 \in v_0 + W, \tag{7}$$

where v_0 in an integer s-dimensional vector and W is an integer cone in $\mathbb{Z}^s$. It gives a reduction of the WWHP to the IHP. This completes the proof of lemma 2.

2 Decidable Variations of $P_{\mathbb{B}}$-Realizability Problem

It turns out that $I_{\mathbb{B}}$- (resp. $S_{\mathbb{B}}$-)realizability is reducible to a restricted version of the integer cone hitting problem. Decidability of this restricted IHP follows from specific properties of maps Φ defined in Section 1.

We use the componentwise partial order on the integer orthant $\mathbb{Z}^d_+$

$$x \prec y \iff x_i \le y_i \text{ for all } i. \tag{8}$$

Up-hitting Problem
INPUT: a square matrix Φ of order d; a d-dimensional vector x_0; a family of vectors $v_i \in \mathbb{Z}^d$, $i = 0, 1, \ldots, r$.
OUTPUT: 'yes' if the *orbit up-shadow* intersects the translate of the integer cone $v_0 + \mathbb{N}(v_1, \ldots, v_r) = v_0 + W$ and 'no' otherwise. It means that

$$\Phi^n x_0 \prec y \tag{9}$$

holds for some integer n and $y \in v_0 + W$.

The **Down-hitting problem** is defined similarly except for the condition (9) which is replaced by the condition

$$y \prec \Phi^n x_0. \tag{10}$$

Lemma 4. *The $I_{\mathbb{B}}$-realizability problem is Turing reducible to the down-hitting problem.*

Proof. Repeat the arguments from the proof of Lemma 1. Now it is possible that some binary words in a block word are missed. It means that the condition (6) is replaced by

$$w(\tau) \prec \Phi^n x_0. \tag{11}$$

Applying the arguments from Lemma 2 we see that (11) is transformed to (10) for cones appeared in the reduction from Lemma 2. □

In a similar way we reduce the surjective filter.

Lemma 5. *The $S_{\mathbb{B}}$-realizability problem is Turing reducible to the up-hitting problem.*

Proof. In a block word taken from the $S_{\mathbb{B}}$ all binary words of the length n appear (possibly, several times). It leads to the condition (9) for cones appeared in the reduction from Lemma 2. □

Dickson's lemma claims that there are no infinite antichains in the poset $(\mathbb{Z}_+^d, \prec)$. So an orbit up-shadow is a finite union of translated copies of the orthant $\mathbb{Z}_+^d$. In the case of an orbit down-shadow copies of the orthant are replaced by 'parallelepipedons' (the Cartesian products of segments). Thus the up-hitting problem as well as the down-hitting problem is reduced to the nonemptiness check for intersections of integer cones, which is an integer linear programming problem, provided the representations mentioned above can be constructed algorithmically.

To construct the aforementioned representations we use specific properties of asymptotic behavior of the orbit points $\Phi^n x_0$, where Φ is determined by (4).

Recall that an $e(g)$-component of $\Phi^n x_0$ is expressed as the number $\nu_n(g)$ of walks of the length n from the vertex id to the vertex g in the graph Γ. The vertex set of the graph Γ is $V(\Gamma) = Q^Q$ and the edge set $E(\Gamma)$ consists of pairs in the form (f, ff_0) or (f, ff_1). (See Section 1 for a detailed exposition.)

Note that the integer $\nu_n(g)$ is the number of words of the length n in a regular language. This language is recognized by an automaton with the transition graph Γ. Thus the generating function $\varphi_g(t) = \sum_{n=0}^{\infty} \nu_n(g)t^n$ for the language is a rational function and its representation in the form $P(t)/Q(t)$ can be found algorithmically.

In the arguments below we need some properties specific to generating functions of regular languages and it is more suitable to analyze the asymptotic behavior in combinatorial settings by considering walks on the graph Γ.

Proposition 1. *For any vertex $g \in V(\Gamma)$ the set $P_g = \{n : \nu_n(g) > 0\}$ is a semilinear set (a finite union of an exceptional finite set and finite collection of arithmetic progressions) and its description can be constructed algorithmically.*

Proof. Regard the Γ as the transition graph of a nondeterministic automaton over a 1-letter alphabet. Now the proposition follows from Parikh's theorem. □

Remark 3. Differences of all progressions in Proposition 1 are the cycle lengths in the graph Γ. So they are divisors of the least common multiple of integers from 1 to $|V(\Gamma)|$. In the sequel we denote this common multiple by N.

Take a vertex g on a directed cycle of the length ℓ. The following inequality

$$\nu_{n+\ell}(g) \geq \nu_n(g). \tag{12}$$

holds. Indeed, one can extend any walk of the length n by the cycle.

Now we divide the vertices of the graph Γ into three groups.

– V_1 consists of vertices v such that some directed cycle (possibly, a loop) goes through the vertex v.
– V_2 consists of vertices v such that there is a walk starting at the id, finishing at the v and passing through a vertex from the set V_1.
– V_3 consists of all other vertices.

Proposition 2. *If $g \in V_3$ then $\nu_n(g) = 0$ for $n > |V(\Gamma)|$.*

Proof. Any walk of the length $n > |V(\Gamma)|$ from id to g must contain repeating vertices. It means that a part of the walk is a directed cycle. So the walk passes a vertex from the set V_1. □

Proposition 3. *The inequality*

$$\nu_{n+N}(g) \geq \nu_n(g), \tag{13}$$

where $N = \mathrm{LCM}(1, \ldots, |V(\Gamma)|)$, holds for all $g \in V(\Gamma)$ and $n > |V(\Gamma)|$.

Proof. For $g \in V_3$ apply Proposition 2.

For $g \in V_1$ the inequality (13) follows from (12) and Remark 3.

Now take a vertex $g \in V_2$. The set $V_g \subseteq V_1$ consists of vertices $g' \in V_1$ such that g' belongs to a walk from id to g and for all walks of this type all vertices after the g' along a walk are in the set V_2. The subgraph Γ_g is induced by the edges of all walks from a vertex in V_g to g.

Let observe the following properties of the Γ_g.

By definition of V_g no edges of Γ_g point to vertices of V_g.

There are no edges outgoing from the vertex g in the graph Γ_g. Otherwise one would detect a directed cycle passing through g.

From these properties we conclude that the graph Γ_g is acyclic. By definition there are no directed cycles passing through vertices in the set V_2. Other vertices in the Γ_g are in the set V_g. There are no directed cycles passing through these vertices because there are no edge ingoing to them.

Note also that the maximum of the path length from $g' \in V_g$ to g does not exceed $|V(\Gamma)|$.

From all the properties above we get

$$\nu_n(g) = \sum_{\substack{g' \in V_g \\ k \leq |V(\Gamma)|}} p_{g',k} \nu_{n-k}(g'), \tag{14}$$

where $p_{g',k}$ is the number of paths from g' and g in the Γ_g.

Applying the inequality (13) to all terms in the right-hand side of (14) we get the same inequality for the left-hand side, i.e. for the vertex g. □

Theorem 2. *The $S_{\mathbb{B}}$-realizability problem is decidable.*

Proof. It follows from Lemma 5 that it is enough to construct an integer cone representation for an orbit up-shadow. Proposition 3 implies that the orbit up-shadow is $\bigcup_{i=0}^{N} \left(\Phi^i x_0 + \mathbb{N}^m\right)$, where m is a dimension, i.e. the cardinality of Q^Q. So the problem is reduced to the integer linear programming problem. □

To prove decidability of the injective filter we should determine for each $0 \leq r < N$ unbounded components of $\Phi^{nN+r}x_0$ and the limit values of the bounded components.

All subsequences $t_g^r(n) = \nu_{nN+r}(g)$, where $0 \leq r < N$, are nondecreasing due to Proposition 3.

A subsequence $t_g^r(n)$, where $g \in V_3$ stabilizes for $n > |V(\Gamma)|$ and the limit value for it is 0.

It follows from (14) that a subsequence $t_g^r(n)$, where $g \in V_2$, tends to infinity iff at least one of the subsequences $t_{g'}^{r-k}(n)$ tends to infinity, where $p_{g',k} \neq 0$.

The remaining case $t_g^r(n)$, where $g \in V_1$, is covered by the following proposition.

Proposition 4. *Let $g \in V_1$. Then $\lim\limits_{n\to\infty} t_g^r(n) = \infty$ iff there exist a directed cycle C passing through g and an edge (g', g'') such that*

(i) *the edge (g', g'') is not included in the cycle C;*
(ii) *the cycle C passes through the vertex g'';*
(iii) $\nu_{nN+r-\ell-1}(g') > 0$, *where ℓ is the distance from g'' to g along the cycle C.*

Note that due to Remark 3 the conditions (iii) are equivalent for all n. It is clear that (i)–(iii) can be verified algorithmically.

Proof. The 'if' part of the proposition follows from

$$\nu_{(n+1)N+r}(g) \geq \nu_{nN+r}(g) + \nu_{(n+1)N+r-\ell-1}(g') > \nu_{nN+r}(g). \tag{15}$$

To prove the 'only if' part suppose that $\nu_{nN+r}(g) = T$ for $n > n_0$. For any directed cycle C passing through g and any edge satisfying (i)–(ii) the first inequality in (15) implies that $\nu_{nN+r-\ell-1}(g') = 0$ for $n > n_0$. □

Theorem 3. *The $I_{\mathbb{B}}$-realizability problem is decidable.*

Proof. Determine all unbounded components for all subsequences $\Phi^{nN+r}x_0$ and the limit values for bounded components. For this purpose use Proposition 4 and the observations preceding it. Then the down-shadow is the union of the sets

$$\begin{cases} y_g \geq 0, & \text{if } \lim\limits_{n\to\infty} (\Phi^{nN+r}x_0)_g = \infty, \\ y_g^\infty \geq y_g \geq 0, & \text{if } \lim\limits_{n\to\infty} (\Phi^{nN+r}x_0)_g = y_g^\infty. \end{cases}$$

over all $0 \leq r < N$. Here y_g are coordinates in the space $\mathbb{Q}^{Q^Q}$.

So the problem is reduced to the integer linear programming problem. □

3 An Undecidable Problem

Theorem 4. *There are alphabets Σ_1, Σ_2 such that the* $(\mathrm{Per}_{\Sigma_1} \| P_{\Sigma_2})$*-realizability problem is undecidable.*

A suitable undecidable problem that is reduced to the $(\mathrm{Per}_{\Sigma_1} \| P_{\Sigma_2})$-realizability problem is the following.

Zero in the Upper Right Corner Problem. (The ZURC problem.) For a given collection of $D \times D$ integer matrices $A_1, \ldots, A_N$ check whether the multiplicative semigroup generated by $\{A_i\}$ contains a matrix M such that $M_{1D} = 0$.

In other words the problem is to check the existence of an integer sequence $j_1, \ldots, j_\ell$, where $1 \leq j_t \leq N$, such that

$$(A_{j_1} A_{j_2} \ldots A_{j_\ell})_{1D} = 0. \tag{16}$$

Theorem 5 ([1]). *The ZURC problem is undecidable for $N = 2$ and $D = 18$.*

We will reduce the ZURC problem with $N = 2$, $D = 18$ to the $(\mathrm{Per}_{\Sigma_1} \| P_{\Sigma_2})$-realizability problem, where

$$\Sigma_1 = [1, \ldots, N], \qquad \Sigma_2 = [1, \ldots, D] \times [1, \ldots, D] \times \{0, 1\}. \tag{17}$$

The reduction is similar to the reduction in [5, Section 4].

Let $A_1, \ldots, A_N$ be an instance of the ZURC problem for $D = 18$, $N = 2$. Rewrite the matrices in the form

$$A_j = \begin{pmatrix} \varepsilon^j_{11} m^j_{11} & \varepsilon^j_{12} m^j_{12} & \ldots & \varepsilon^j_{1D} m^j_{1D} \\ \varepsilon^j_{21} m^j_{21} & \varepsilon^j_{22} m^j_{22} & \ldots & \varepsilon^j_{2D} m^j_{2D} \\ \ldots & \ldots & \ldots & \ldots \\ \varepsilon^j_{D1} m^j_{D1} & \ldots & \varepsilon^j_{D\,(D-1)} m^j_{D\,(D-1)} & \varepsilon^j_{DD} m^j_{DD} \end{pmatrix},$$

where $m^j_{ik} > 0$ and $\varepsilon^j_{ik} \in \{\pm 1, 0\}$. Let M be the maximum of m^j_{ik}.

Fix now a sequence A_{j_1}, A_{j_2}, $\ldots$, A_{j_ℓ}. Matrix elements in the product $A_{j_1} A_{j_2} \ldots A_{j_\ell}$ have the form

$$\big(A_{j_1} A_{j_2} \ldots A_{j_\ell}\big)_{ik} = \sum_\tau \varepsilon(\tau) m(\tau), \tag{18}$$

where τ runs over all sequences of pairs $(i_\alpha k_\alpha)$ such that the length of a sequence is ℓ and $i_1 = i$, $k_\ell = k$, $i_{\alpha+1} = k_\alpha$,

$$\varepsilon(\tau) = \prod_{\alpha=1}^{\ell} \varepsilon^{j_\alpha}_{i_\alpha k_\alpha}, \qquad m(\tau) = \prod_{\alpha=1}^{\ell} m^{j_\alpha}_{i_\alpha k_\alpha}. \tag{19}$$

Using the expansion (18, 19) we define the partition of words of the length $\ell \cdot \lceil \log_2 M \rceil$ over the alphabet $\Sigma_1 \times \Sigma_2$ into three sets $T^+_{ik}(\boldsymbol{j})$, $T^-_{ik}(\boldsymbol{j})$ and $T^{\mathrm{bad}}_{ik}(\boldsymbol{j})$, where $\boldsymbol{j} = j_1, \ldots, j_\ell$. (Below we drop out $\boldsymbol{j}$ while the sequence $\boldsymbol{j}$ is fixed.)

It is convenient to represent a word over the alphabet $\Sigma_1 \times \Sigma_2$, where Σ_i are given by (17), by a 4-row table. A table column represents a symbol in the word. The first row bears symbols from Σ_1 while the remaining three columns represent symbols from Σ_2 (they have three components as indicated in (17)).

A word in the set T^+_{ik} (T^-_{ik}) can be divided in ℓ subwords of the length $\lceil \log_2 M \rceil$. The α-th subword has the form

$$\begin{pmatrix} j_\alpha & j_\alpha & \dots & j_\alpha \\ i_\alpha & i_\alpha & \dots & i_\alpha \\ k_\alpha & k_\alpha & \dots & k_\alpha \\ \beta_0 & \beta_1 & \dots & \beta_{\lceil \log_2 M \rceil - 1} \end{pmatrix} . \tag{20}$$

in the table representation defined above. The upper three elements in each row are the same in (20). Note that j_α is the α-th element of the sequence $\boldsymbol{j}$. The fourth row is a binary representation of an integer β.

A word in the set T^+_{ik} should satisfy the following requirements (a) $i_1 = i$; (b) $k_\ell = k$; (c) $i_{\alpha+1} = k_\alpha$; (d) $\beta < m^{j_\alpha}_{i_\alpha k_\alpha}$; (e) $\varepsilon(\tau) = 1$, where τ is the sequence of pairs (i_α, k_α) and $\varepsilon(\tau)$ is given by (19).

A word in the set T^-_{ik} should satisfy the requirements (a)–(d) and the modified requirement (e′) $\varepsilon(\tau) = -1$.

The set T^{bad}_{ik} collects the rest of words.

Proposition 5. *In the notation above* $\left(A_{j_1} A_{j_2} \dots A_{j_\ell}\right)_{ik} = |T^+_{ik}| - |T^-_{ik}|$.

Proof. Restricting words in the set T^+_{ik} to the upper three rows one obtains all correct sequences τ such that $\varepsilon(\tau) = 1$. The same restriction for the set T^-_{ik} gives the sequences τ such that $\varepsilon(\tau) = -1$.

The multiplicity of a sequence τ in the set $T^\pm_{ik}$ depends on fourth rows of the tables. According to the requirement (d) and the permutation property there are exactly $m^{j_\alpha}_{i_\alpha k_\alpha}$ subwords bearing $(j_\alpha, i_\alpha, k_\alpha)$ in the upper three rows. So the multiplicity of the sequence τ is $m(\tau)$, where $m(\tau)$ is given by (19). □

It follows from the above that the sets $T^\$_{ik}(\boldsymbol{j})$, where $\$ \in \{+, -, \text{bad}\}$, are disjoint for different $\boldsymbol{j}$ because the sequence $\boldsymbol{j}$ is recovered from the first row in the table representation.

Proposition 6. *For a fixed collection of matrices $A_1, \dots, A_N$ and for each pair i, k the language $T^{\text{all}, \$}_{ik} = \bigcup_{\boldsymbol{j}} T^\$_{ik}(\boldsymbol{j})$ is regular for $\$ \in \{+, -, \text{bad}\}$.* □

Proof. The requirements (a)–(d) are verified by local check on the subwords of the fixed length $\lceil \log_2 M \rceil$.

Computing the sign $\varepsilon(\tau)$ can be done in the cyclic group of two elements.

So the sets $T^{\text{all}, \pm}_{ik}$ are regular. But the class of regular languages is closed under the complement. Thus the set $T^{\text{all}, \text{bad}}_{ik}$ is also regular. □

Sketch of proof of Theorem 4. We repeat the construction from [5, Section 4]. The reduction algorithm takes an instance of the ZURC problem ($N = 2$, $D = 18$)

and constructs an automaton C such that the condition (16) holds for some element in the semigroup generated by the input matrices if and only if $L(C) \cap \mathrm{Per}_{\Sigma_1} \| P_{\Sigma_2} \neq \varnothing$.

The automaton expects an input w from the language $\mathrm{Per}_{\Sigma_1} \| P_{\Sigma_2}$ such that block rank is $\ell \cdot \lceil \log_2 M \rceil$ and the word w is a certificate for zero representation (16). A period in the first (periodic) component determines the sequence $\boldsymbol{j} = j_1, \dots, j_\ell$ such that (16) holds. The automaton expects that each symbol in the sequence $\boldsymbol{j}$ is repeated $\lceil \log_2 M \rceil$ times. In the second (permutation) component the automaton expects that the blocks from the sets $T^+_{1D}(\boldsymbol{j})$, $T^-_{1D}(\boldsymbol{j})$ are paired and are followed by the blocks from the set T^{bad}_{1D}. Note that such a pairing exists iff $|T^+_{1D}| = |T^-_{1D}|$.

The correctness of the reduction is proved in a way similar to the arguments in [5, Section 4]. If the automaton accepts a word w in the language $\mathrm{Per}_{\Sigma_1} \| P_{\Sigma_2}$ then one can extract the sequence $\boldsymbol{j}$ from the first components of symbols in the word w. The check in the second components guarantees that (16) holds for the sequence $\boldsymbol{j}$. We apply here Proposition 5.

In other direction, if (16) holds for a sequence $\boldsymbol{j}$ then there exists a word in the language $\mathrm{Per}_{\Sigma_1} \| P_{\Sigma_2}$ satisfying the properties expected (and verified) by the automaton C. Thus $L(C) \cap \mathrm{Per}_{\Sigma_1} \| P_{\Sigma_2} \neq \varnothing$.

Remark 4. By suitable encoding the symbols of the alphabets Σ_1 and Σ_2 one can prove that the $(\mathrm{Per}_{\mathbb{B}} \| P_{\mathbb{B}})$-realizability problem is also undecidable.

References

1. Bell, P., Potapov, I.: Lowering Undecidability Bounds for Decision Questions in Matrices. In: Ibarra, O.H., Dang, Z. (eds.) DLT 2006. LNCS, vol. 4036, pp. 375–385. Springer, Heidelberg (2006)
2. Blondel, V.D., Portier, N.: The presence of a zero in an integer linear recurrent sequence is NP-hard to decide. Linear Algebra and Its Applications 351-352, 91–98 (2002)
3. Halava, V., Harju, T., Hirvensalo, M., Karhumäki, J.: Skolem's problem — on the border between decidability and undecidability. TUCS Tech. Rep. No 683 (2005)
4. Parikh, R.J.: On context-free languages. Journal of the ACM 13(4), 570–581 (1966)
5. Tarasov, S., Vyalyi, M.: Orbits of linear maps and regular languages, http://arxiv.org/abs/1011.1842
6. Vereshchagin, N.K.: On occurrence of zero in a linear recurrent sequence. Math. notes 38(2), 177–189 (1985)
7. Vyalyi, M.N.: On models of a nondeterministic computation. In: Frid, A., Morozov, A., Rybalchenko, A., Wagner, K.W. (eds.) CSR 2009. LNCS, vol. 5675, pp. 334–345. Springer, Heidelberg (2009)
8. Vyalyi, M., Tarasov, S.: Orbits of linear maps and regular languages. Discrete Analysis and Operations Research 17(6), 20–49 (2010)

Shared-Memory Systems and Charts*

Rémi Morin

Laboratoire d'Informatique Fondamentale de Marseille;
LIF; CNRS; UMR 6166
and
Aix-Marseille Université, 163, avenue de Luminy,
F-13288 Marseille, France

Abstract. In this paper we extend several results from Mazurkiewicz trace theory to the framework of cut-bounded languages. First the expressive power of shared-memory systems in terms of recognized sets of labeled partial orders is characterized by means of the notion of cut-bound plus the condition to be regular, or equivalently MSO-definable. Next weakly unambiguous systems with deterministic rules are proved to be as expressive as any system, contrary to deterministic or strongly unambiguous systems considered previously. Finally we extend the rational description of regular trace languages by loop-connected specifications within a new algebraic framework called shared-memory charts. In that way we present also several generalizations of results from the theory of regular sets of message sequence charts.

Introduction

Mazurkiewicz traces are a well-known and intensively studied class of labeled partial orders which benefits from a rich theory and also still open problems [10,18,22]. In particular, they are closely related to the semantics of distributed devices, called asynchronous automata, which communicate by means of synchronizations [23]. For the communication paradigm based on message passing, an analogous theory has been developed in the past years for message sequence charts, see e.g. [16]. In this paper we consider systems that communicate by means of shared variables and present a generalization of several key results from the theory of Mazurkiewicz traces and message sequence charts [4,15,16,18,22,23].

As opposed to our previous works [19,20], we consider in this paper shared-memory systems without any restriction, neither determinism [19], nor strong unambiguity [20]. This requires to focus on the notion of chain covering from [12,17], or equivalently the definition of cut-bound. We prove first that a set of labeled partial orders corresponds to the behaviours of a shared-memory system if and only if it is cut-bounded and regular, a notion borrowed from [14]. To do so, we extend the connection between MSO-logic and regularity from Mazurkiewicz traces [22] to cut-bounded languages. We consider also in this paper the notions of deterministic rules and (weak) unambiguity. As opposed to

* This work is partly supported by project ECSPER (ANR-09-JCJC-0069).

deterministic or strongly unambiguous systems, we prove that the language of any shared-memory system is also the language of an unambiguous system with deterministic rules.

In order to describe behaviours of shared-memory systems in some algebraic way, we introduce next the notion of shared-memory charts. Interestingly this framework can be regarded formally as a generalization of Mazurkiewicz traces and message sequence charts. Moreover, similarly to [18], we can characterize the cut-bounded languages that are regular by considering shared-memory chart specifications that are loop-connected. We explain also how to decide whether the language of a given shared-memory chart specification is cut-bounded. To do so we adapt the definition of communication graph [4,16] from message sequence charts to shared-memory charts and next generalize the corresponding criterion for non-divergence [4] to the notion of cut-bound.

Preliminaries. A labeled partial order, partial word, or *pomset* (for partially ordered multiset) over a finite alphabet Σ is a triple $t = (E, \preccurlyeq, \xi)$ where $(E, \preccurlyeq)$ is a finite partial order and ξ is a mapping from E to Σ *without autoconcurrency*: $\xi(x) = \xi(y)$ implies $x \preccurlyeq y$ or $y \preccurlyeq x$ for all $x, y \in E$. As usual we shall not distinguish between isomorphic pomsets. We denote by $\mathbb{P}(\Sigma)$ the class of all pomsets over Σ. A pomset can be seen as an abstraction of an execution of a concurrent system. In this view, the elements x of E are *events* and their label $\xi(x)$ describes the action performed by the system when event x occurs. Moreover the ordering $x \preccurlyeq y$ means that x *happens before* y.

An *ideal* of a pomset $t = (E, \preccurlyeq, \xi)$ is a subset $H \subseteq E$ such that $x \in H$ and $y \preccurlyeq x$ imply $y \in H$. Then the restriction $t|H = (H, \preccurlyeq \cap (H \times H), \xi \cap (H \times \Sigma))$ is called a *prefix* of t and we write $t' \leqslant t$. For all $z \in E$, we denote by $\downarrow z$ the ideal of events below z, i.e. $\downarrow z = \{y \in E \mid y \preccurlyeq z\}$. We write $x \prec\!\!\cdot\, y$ and say that y *covers* x if $x \prec y$ and $x \prec z \preccurlyeq y$ implies $y = z$.

Let us recall some basic notions of Mazurkiewicz trace theory [9,10]. The concurrency of a distributed system is often represented by an *independence relation* over the alphabet of actions Σ, that is a binary, symmetric, and irreflexive relation $\parallel \subseteq \Sigma \times \Sigma$. Then a *Mazurkiewicz trace* over the independence alphabet $(\Sigma, \parallel)$ is a pomset $t = (E, \preccurlyeq, \xi)$ over Σ satisfying the two next requirements:

M_1: For all events $e_1, e_2 \in E$ with $\xi(e_1) \not\parallel \xi(e_2)$, we have $e_1 \preccurlyeq e_2$ or $e_2 \preccurlyeq e_1$;
M_2: For all events $e_1, e_2 \in E$ with $e_1 \prec\!\!\cdot\, e_2$, we have $\xi(e_1) \not\parallel \xi(e_2)$.

We denote by $\mathbb{M}(\Sigma, \parallel)$ the class of all Mazurkiewicz traces over $(\Sigma, \parallel)$.

1 Communication with Shared Variables

Throughout the paper we fix some finite alphabet Σ. The notion of a shared-memory system we consider is based on a finite set $\mathcal{I}$ of processes together with a distributed alphabet $(\Sigma_i)_{i \in \mathcal{I}}$ where $\Sigma_i \subseteq \Sigma$ for every process i. For each action $a \in \Sigma$, the *location* $\text{Loc}(a) = \{i \in \mathcal{I} \mid a \in \Sigma_i\}$ collects all processes involved in action a. Intuitively each occurrence of action a induces a *synchronized* step of all processes from $\text{Loc}(a)$. For that reason we assume that $\text{Loc}(a)$ is *non-empty* for all $a \in \Sigma$, i.e. $\bigcup_{i \in \mathcal{I}} \Sigma_i = \Sigma$.

1.1 Shared-Memory Systems

Processes of a shared-memory system can communicate by means of a finite set $\mathcal{R}$ of shared variables (or registers) taking values from a common finite set of data $\mathcal{D}$; in particular the initial content of this shared memory is described by a *configuration* $\imath : \mathcal{R} \to \mathcal{D}$ that associates to each register $r \in \mathcal{R}$ a value $\imath(r) \in \mathcal{D}$. Intuitively each action corresponds to reading the values of a subset of registers (the guard) and next writing new values in some other registers (the update). For convenience, we shall allow distinct processes to read the value of a register concurrently; but we forbid the writing of a new value in the same register by two different processes simultaneously, that is, we shall consider *Concurrent-Read Exclusive-Write* systems, only. A *valuation* is a partial function $\nu : \mathcal{R} \rightharpoonup \mathcal{D}$; it will correspond to the reading or the writing of some values in a subset of registers. The *domain* $\mathrm{dom}(\nu)$ of a valuation ν is the set of registers r for which $\nu(r)$ is defined. We denote by $\mathcal{V}$ the set of all valuations. Given a configuration $q : \mathcal{R} \to \mathcal{D}$ and a subset of registers $R \subseteq \mathcal{R}$, we let $q|R$ denote the valuation with domain R such that $q|R(r) = q(r)$ for all $r \in R$. We denote by Q the set $\mathcal{D}^{\mathcal{R}}$ of all configurations. A *rule* is a triple (ν, a, ν') where $a \in \Sigma$ and $\nu, \nu' \in \mathcal{V}$ are two valuations.

DEFINITION 1.1. *A* shared-memory system *(for short, an SMS) consists of a set of rules $\Delta \subseteq \mathcal{V} \times \Sigma \times \mathcal{V}$, some initial configuration $\imath : \mathcal{R} \to \mathcal{D}$ and a subset $F \subseteq Q$ of final configurations.*

We stress here that *we consider in this paper only systems over finite alphabets with finitely many registers and data.* Intuitively action a can occur synchronously on all processes from $\mathrm{Loc}(a)$ in some configuration q if there exists some rule $(\nu, a, \nu') \in \Delta$ such that $\nu = q|\mathrm{dom}(\nu)$, i.e. the guard ν is satisfied. In that case processes from $\mathrm{Loc}(a)$ may write the new values $\nu'(r)$ in registers from the domain of ν'. The step consisting of all these readings and all these writings is considered atomic. For convenience we put $\rho = (\nu_\rho, a_\rho, \nu'_\rho)$, $\mathrm{R}_\rho = \mathrm{dom}(\nu_\rho)$ and $\mathrm{W}_\rho = \mathrm{dom}(\nu'_\rho)$ for each rule $\rho \in \Delta$. A rule ρ is *enabled* in the configuration q if $q|\mathrm{R}_\rho = \nu_\rho$.

1.2 Partial-Order Semantics of Shared-Memory Systems

Let $\rho, \rho' \in \Delta$ be two rules. We put $\rho \| \rho'$, and we say that ρ and ρ' are independent, if $\mathrm{Loc}(a_\rho) \cap \mathrm{Loc}(a_{\rho'}) = \emptyset$, $\mathrm{W}_\rho \cap (\mathrm{R}_{\rho'} \cup \mathrm{W}_{\rho'}) = \emptyset$, and $\mathrm{W}_{\rho'} \cap (\mathrm{R}_\rho \cup \mathrm{W}_\rho) = \emptyset$. Thus two rules are independent if they correspond to actions performed by disjoint sets of processes and if each rule does not modify the registers read or written by the other. These conditions are known as Bernstein's conditions [5]. Let $t = (E, \preccurlyeq, \xi)$ be a pomset over the set of actions Σ. In order to reason about which registers are read by each event of t and how events change the values of registers, we make use of the notion of run. A *run* of t over the SMS $\mathcal{S}$ is a mapping $\rho : E \to \Delta$ which maps each event $e \in E$ to some rule $\rho(e) \in \Delta$ such that $a_{\rho(e)} = \xi(e)$, which means simply that the rule $\rho(e)$ corresponds to the action $\xi(e)$ performed by e. We say that a run ρ of $t = (E, \preccurlyeq, \xi)$ is *valid* if the following conditions are satisfied:

V_1: For all events $e_1, e_2 \in E$ with $\rho(e_1) \not\parallel \rho(e_2)$, we have $e_1 \preccurlyeq e_2$ or $e_2 \preccurlyeq e_1$;
V_2: For all events $e_1, e_2 \in E$ with $e_1 \prec\!\cdot\, e_2$, we have $\rho(e_1) \not\parallel \rho(e_2)$.

The first condition ensures that two dependent rules cannot occur concurrently. In particular if e and e' are two events that change the value of some register r then e and e' must be comparable w.r.t. $\preccurlyeq$. The second condition asserts that each covering relation between two events results from some ordering constraint. We distinguish three cases in this situation. First the two rules $\rho(e_1)$ and $\rho(e_2)$ occur on a common process or write a new value in a common register. Second $\rho(e_1)$ writes a value in some register and this value is expected by the guard of $\rho(e_2)$. Third $\rho(e_1)$ reads a register that is rewritten by $\rho(e_2)$. Observe here that V_1 and V_2 amount to require that the triple $(E, \preccurlyeq, \rho)$ is a Mazurkiewicz trace from $\mathbb{M}(\Delta, \parallel)$.

We assume now that ρ is a valid run for t. If t describes a concurrent computation of the SMS $\mathcal{S}$ where each event e applies rule $\rho(e)$ then any prefix of t should correspond to some partial computation of $\mathcal{S}$. Let $H \subseteq E$ be an ideal of t. The configuration $q_{\rho,H}$ after H corresponds intuitively to a snapshot of the system when all events of H have occurred along the execution of t w.r.t. ρ: The value of each register is the value written by the last event that has modified this value. Formally $q_{\rho,H}$ is the configuration such that for all registers $r \in \mathcal{R}$, $q_{\rho,H}(r) = \nu'_{\rho(e)}(r)$ if e is the greatest event in H such that $r \in \mathrm{W}_{\rho(e)}$, and $q_{\rho,H}(r) = \imath(r)$ if there is no such event. A valid run ρ of t is *applicable in the SMS* $\mathcal{S}$ if the rule $\rho(e)$ is enabled in the configuration $q_{\rho,\downarrow e\setminus\{e\}}$ for all events $e \in E$. Moreover an applicable run of $t = (E, \preccurlyeq, \xi)$ is *accepting* if $q_{\rho,E} \in F$.

Definition 1.2. *The language $\mathcal{L}(\mathcal{S})$ recognized by $\mathcal{S}$ collects all pomsets which admit an accepting run.*

1.3 Asynchronous Automata and Zielonka's Theorem

The notion of *asynchronous automata* originates from the seminal work by Zielonka [23]. The definition adopted in [10, chap. 7] can be identified with a subclass of shared-memory systems such that $\mathcal{I} = \mathcal{R}$, which means intuitively that each register corresponds to the local state of a process. In this setting each action is assigned a read domain $\mathrm{R}_a \subseteq \mathcal{R}$ and a write domain $\mathrm{W}_a \subseteq \mathcal{R}$ such that $\mathrm{Loc}(a) = \mathrm{W}_a \subseteq \mathrm{R}_a$. It is required that for any rule $\rho \in \Delta$ with $a_\rho = a$ we have $\mathrm{R}_\rho = \mathrm{R}_a$ and $\mathrm{W}_\rho = \mathrm{W}_a$. Then the set of rules associated with a can be regarded as a subset $\delta_a \subseteq \mathcal{D}^{\mathrm{R}_a} \times \mathcal{D}^{\mathrm{W}_a}$. Now we can define an independence relation $\parallel_\Sigma$ over Σ such that $a \parallel_\Sigma b$ if $\mathrm{W}_a \cap (\mathrm{R}_b \cup \mathrm{W}_b) = \emptyset$ and $\mathrm{W}_b \cap (\mathrm{R}_a \cup \mathrm{W}_a) = \emptyset$. These conditions imply that $\mathrm{Loc}(a) \cap \mathrm{Loc}(b) = \emptyset$. Then the language recognized by such an asynchronous automaton according to Definition 1.2 consists of Mazurkiewicz traces over $(\Sigma, \parallel_\Sigma)$. This language coincides with the usual semantics of asynchronous automata.

Consider now a finite independence alphabet $(\Sigma, \parallel)$. A trace language $\mathcal{L} \subseteq \mathbb{M}(\Sigma, \parallel)$ is called *regular* if the collection of all linear extensions is a regular word language. Zielonka's theorem captures this notion of regularity with the help of asynchronous automata as follows.

Theorem 1.3. *[23] Let $\mathcal{L} \subseteq \mathbb{M}(\Sigma, \|)$ be a set of Mazurkiewicz traces over some finite independence alphabet $(\Sigma, \|)$. Then $\mathcal{L}$ is regular if and only if it is recognized by some asynchronous automaton.*

2 Expressive Power of Shared-Memory Systems

The first goal of this work is to extend Theorem 1.3 to the general setting of shared-memory systems. To do so we need to introduce the notion of cut-bound.

2.1 Cut-Bounded Languages

Definition 2.1. *Let $t = (E, \preccurlyeq, \xi)$ be a pomset over Σ and $t' = (E', \preccurlyeq', \xi')$ be a prefix of t. The* cut-width *of the pair (t', t) is the number of events from E' that are covered by some event from $E \setminus E'$ in t. The* cut-bound *of a language $\mathcal{L} \subseteq \mathbb{P}(\Sigma)$ is the least upper bound $B \in \mathbb{N} \cup \{\infty\}$ of the cut-widths of all pairs (t', t) where $t' \leqslant t$ and $t \in \mathcal{L}$. A language is* cut-bounded *if its cut-bound is finite.*

Example 2.2. Consider the alphabet $\Sigma = \{a, b\}$. A *ladder* is a pomset over Σ that consists of a chain of n a-events and a chain of n b-events and such that the k^{th} b covers the k^{th} a and no b is below any a. An example of a ladder is depicted in Figure 1. Clearly the set of all ladders is *not* cut-bounded.

It is easy to see that the set of all Mazurkiewicz traces over some finite independence alphabet $(\Sigma, \|)$ is cut-bounded with cut-bound at most $|\Sigma|^2$. Consider now an SMS $\mathcal{S}$. Each concurrent computation $t = (E, \preccurlyeq, \xi)$ from $\mathcal{L}(\mathcal{S})$ corresponds to an accepting run $\rho : E \to \Delta$ (where Δ denotes the set of rules) and some Mazurkiewicz trace $t^\circ = (E, \preccurlyeq, \rho)$. Therefore $\mathcal{L}(\mathcal{S})$ is cut-bounded and its cut-bound is at most $|\Delta|^2$. Thus,

Lemma 2.3. *The language of any shared-memory system is cut-bounded.*

In this section we shall characterize which cut-bounded languages are recognized by shared-memory systems. Before that, it is interesting to relate the class of cut-bounded languages to an equivalent notion introduced by Kuske.

Definition 2.4. *[17] Let $k \in \mathbb{N}$ and $t = (E, \preccurlyeq, \xi)$ be a pomset over Σ. A family $(C_i)_{i \in [1,k]}$ of subsets of E is called a k-*chain covering *of t if*

1. *each C_i is a chain in $(E, \preccurlyeq)$, i.e. $(C_i, \preccurlyeq \cap (C_i \times C_i))$ is a linear order;*
2. *$E = \bigcup_{i \in [1,k]} C_i$;*
3. *$\forall e, f \in E$: $(e \prec\!\!\cdot f \Rightarrow \exists i \in [1, k], \{e, f\} \subseteq C_i)$.*

We let $\mathbb{P}_k(\Sigma)$ collect all pomsets over Σ that admit a k-chain covering.

By Dilworth's theorem [11], the first two requirements are equivalent to saying that the width of $(E, \preccurlyeq)$ is at most k. So the crucial part is the third requirement. It enforces that not only the partial order $(E, \preccurlyeq)$ is covered by the chains, but that in addition the end points of any covering relation belong to some common chain. It is clear that the cut-bound of any pomset language within $\mathbb{P}_k(\Sigma)$ is at most k. Conversely Dilworth's theorem can be used to show that any pomset language with cut-bound at most k is included in $\mathbb{P}_k(\Sigma)$.

2.2 Refinements of Cut-Bounded Languages

Let Σ_1 and Σ_2 be two alphabets and $\pi : \Sigma_1 \to \Sigma_2$ be a mapping from Σ_1 to Σ_2. This mapping extends into a morphism from $\Sigma_1^\star$ to $\Sigma_2^\star$. It extends also in a natural way into a function that maps each pomset $t = (E, \preccurlyeq, \xi)$ over Σ_1 to the triple $\pi(t) = (E, \preccurlyeq, \pi \circ \xi)$. The latter might not be a pomset over Σ_2 in case some autoconcurrency appears in it (see preliminaries). This situation occurs if $\pi(a) = \pi(b)$ for two distinct actions $a, b \in \Sigma$ while there are two events in t that are not comparable and that are labelled by a and b.

Definition 2.5. *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be two sets of pomsets over Σ_1 and Σ_2 respectively and $\pi : \Sigma_1 \to \Sigma_2$ be a mapping from Σ_1 to Σ_2. Then π is a* refinement *from $\mathcal{L}_2$ onto $\mathcal{L}_1$ if $\pi(\mathcal{L}_1) = \mathcal{L}_2$ and $\pi : \mathcal{L}_1 \to \mathcal{L}_2$ is a bijection.*

The condition $\pi(\mathcal{L}_1) = \mathcal{L}_2$ implies that $\pi(t)$ shows no autoconcurrency for any $t \in \mathcal{L}_1$. It implies also that $\mathcal{L}_1$ and $\mathcal{L}_2$ have the same cut-bound. This definition of refinement is weaker than the notion of *strong refinement* from [19,20] which requires additionally that π induces a bijection from $\text{Pref}(\mathcal{L}_1)$ onto $\text{Pref}(\mathcal{L}_2)$. We stress that we keep here the requirement that π induces a bijection from $\mathcal{L}_1$ onto $\mathcal{L}_2$ in order to be able to apply our results to the restricted case of unambiguous shared-memory systems with deterministic rules at the end of this section.

Let $\mathcal{L}$ be a pomset language over Σ with cut-bound $B < \infty$ and $t = (E, \preccurlyeq, \xi)$ be a pomset from $\mathcal{L}$. We consider two actions $a, b \in \Sigma$. For each cover $e_a \prec\!\cdot\, e_b$ with $\xi(e_a) = a$ and $\xi(e_b) = b$ we denote by $|e_a, e_b|_{a,b}$ the number of covers $e'_a \prec\!\cdot\, e'_b$ with $\xi(e'_a) = a$, $\xi(e'_b) = b$, and $e'_a \preccurlyeq e_a$. Note that these conditions imply that $e'_b \preccurlyeq e_b$. We denote by Γ the set of all partial functions from Σ to $[0, B-1]$ and we write $\gamma(x) = \perp$ if $\gamma \in \Gamma$, $x \in \Sigma$, and $\gamma(x)$ is undefined. We construct a new labeling $\xi^\circ : E \to \Sigma \times \Gamma \times \Gamma$ such that for each event $e \in E$ we have $\xi^\circ(e) = (\xi(e), \gamma, \gamma')$ where for all $x \in \Sigma$:

- $\gamma(x) = |e, f|_{\xi(e),x} \mod B$ if there exists some event f such that $e \prec\!\cdot\, f$ and $\xi(f) = x$, and $\gamma(x) = \perp$ otherwise.
- $\gamma'(x) = |f, e|_{x,\xi(e)} \mod B$ if there exists some event f such that $f \prec\!\cdot\, e$ and $\xi(f) = x$, and $\gamma'(x) = \perp$ otherwise.

Note that in both cases f is unique if it exists, because t shows no autoconcurrency. We denote by t° the pomset $t^\circ = (E, \preccurlyeq, \xi^\circ)$. We provide now the new alphabet $\Sigma^\circ = \Sigma \times \Gamma \times \Gamma$ with some independence relation $\|^\circ$ such that $(a, \gamma_a, \gamma'_a) \not\|^\circ (b, \gamma_b, \gamma'_b)$ if $a = b$, $\gamma_a(b) = \gamma'_b(a) \neq \perp$ or $\gamma_b(a) = \gamma'_a(b) \neq \perp$. Clearly $\|^\circ$ is irreflexive and symmetric.

Lemma 2.6. *For each $t \in \mathcal{L}$, we have $t^\circ \in \mathbb{M}(\Sigma^\circ, \|^\circ)$.*

In that way we have built a refinement $\pi : \Sigma^\circ \to \Sigma$ from $\mathcal{L}$ onto the Mazurkiewicz trace language $\mathcal{L}^\circ = \{t^\circ \mid t \in \mathcal{L}\}$ over the finite independence alphabet $(\Sigma^\circ, \|^\circ)$.

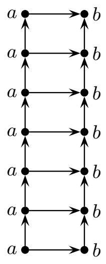


Fig. 1. A ladder

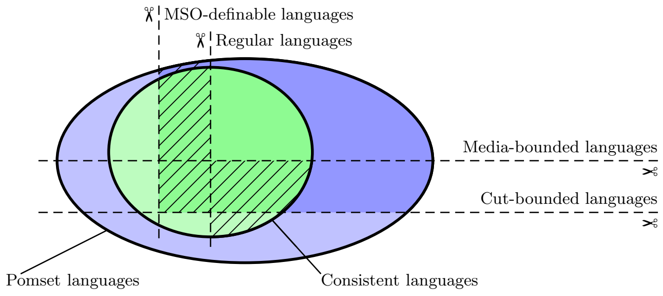


Fig. 2. Classification of cut-bounded languages

2.3 Büchi's Theorem for Cut-Bounded Languages

In language theory, a set of words $L \subseteq \Sigma^\star$ is called regular if it has finitely many residuals. For a given word u, the residual at u consists of all words v such that $u.v \in L$. In particular, if u is not a prefix of some word from L then the residual at u is empty. By analogy with these classical definitions, a definition of regularity for pomset languages was introduced in [14] in order to extend this classical notion.

Let $\mathcal{L}$ be a pomset language and $t_1 = (E_1, \preccurlyeq_1, \xi_1)$ be a pomset over Σ. The *residual* $\mathcal{L} \setminus t_1$ consists of all pomsets $t_2 = (E_2, \preccurlyeq_2, \xi_2)$ such that there exists some pomset $t = (E, \preccurlyeq, \xi)$ in $\mathcal{L}$ satisfying the following conditions:

1. $E = E_1 \cup E_2$, $E_1 \cap E_2 = \emptyset$, and E_1 is an ideal of t,
2. t_1 is the restriction of t to events in E_1, and
3. t_2 is the restriction of t to events in E_2.

Definition 2.7. *Let $\mathcal{L}$ be a set of pomsets. Given two pomsets t and t', we put $t \equiv^r t'$ if $\mathcal{L} \setminus t = \mathcal{L} \setminus t'$. Then $\mathcal{L}$ is* regular *if the equivalence relation $\equiv^r$ is of finite index.*

As observed in [14], this notion of regularity coincides with the usual notions of regularity for Mazurkiewicz traces [10], or message sequence charts [16], and more generally consistent sets of pomsets [3].

In his seminal paper [7] Büchi gave a logical characterization of regular word languages by means of MSO logic. Formulae of the MSO logic that we consider involve first-order variables $x, y, z...$ for events and second-order variables $X, Y, Z...$ for sets of events. They are built up from the atomic formulae $P_a(x)$ for $a \in \Sigma$ (which stands for "the event x is labeled by the action a"), $x \preccurlyeq y$, and $x \in X$ by means of the Boolean connectives $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$ and quantifiers $\exists, \forall$ (both for first order and for set variables). We denote by MSO(Σ) the set of all formulae of MSO. Formulae without free variables are called sentences.

The satisfaction relation $\models$ between pomsets and sentences is defined canonically with the understanding that first order variables range over events of E

and second order variables over subsets of E. The set of pomsets which satisfy a sentence φ is denoted by $\mathrm{Mod}(\varphi)$. We say that a set of pomsets $\mathcal{L}$ is MSO-*definable* if there exists a sentence φ such that $\mathcal{L} = \mathrm{Mod}(\varphi)$.

It was shown by Thomas [22] that regularity corresponds to MSO-definability for any set of Mazurkiewicz traces. We extend easily this connection to any cut-bounded language.

Theorem 2.8. *A cut-bounded language is MSO-definable if and only if it is regular.*

Proof. Let $\mathcal{L} \subseteq \mathbb{P}(\Sigma)$ be a cut-bounded language. By Lemma 2.6, there exists some refinement $\pi : \Gamma \to \Sigma$ from $\mathcal{L}$ onto $\mathcal{L}' \subseteq \mathbb{M}(\Gamma, \|)$ where $(\Gamma, \|)$ is a finite independence alphabet. Let $\mathcal{L}'' = \pi^{-1}(\mathcal{L}) \cap \mathbb{M}(\Gamma, \|)$. Then $\pi(\mathcal{L}'') = \mathcal{L}$. Moreover $\mathcal{L}$ is regular (resp. MSO-definable) iff $\mathcal{L}''$ is regular (resp. MSO-definable). The result follows then from [22]. ■

We have already observed in [20, Prop. 1.6] that the language recognized by some SMS is MSO-definable. Since it is cut-bounded (Lemma 2.3), it is also regular. Our first main result establishes the converse property and states the expected generalization of Zielonka's theorem.

2.4 Generalizations of Zielonka's Theorem

Consider now a regular and cut-bounded pomset language $\mathcal{L} \subseteq \mathbb{P}(\Sigma)$. We can apply the construction of Subsection 2.2 and get a Mazurkiewicz trace language $\mathcal{L}^\circ$. The latter is clearly MSO-definable because $\mathcal{L}$ is MSO-definable (Theorem 2.8). It follows that $\mathcal{L}^\circ$ is regular. By means of Theorem 1.3 we get an asynchronous automaton that recognizes $\mathcal{L}^\circ$. We can apply then [20, Lemma 2.6] and get an SMS that recognizes $\mathcal{L}$. More precisely, the proof of [20, Lemma 2.6] assumes that $\pi : \Sigma^\circ \to \Sigma$ is a *strong* refinement from $\mathcal{L}$ onto $\mathcal{L}(\mathcal{S}^\circ)$, but it applies verbatim under the weaker assumption that π is only a refinement. (The only difference is that the weaker refinements that we consider now do not preserve the property of quasi-unambiguity.)

Theorem 2.9. *Let $\mathcal{L} \subseteq \mathbb{P}(\Sigma)$. The following conditions are equivalent:*

(i) *$\mathcal{L}$ is cut-bounded and regular.*
(ii) *$\mathcal{L}$ is recognized by some SMS.*
(iii) *There exists a refinement from $\mathcal{L}$ onto a regular Mazurkiewicz trace language.*

Proof. We have proved above that (i) ⇒ (iii) ⇒ (ii). On the other hand we have already explained why (ii) implies (i). ■

There are three sources of non-determinism in shared-memory systems. Whenever the system executes an action, it first guesses the registers to read, next chooses which registers to modify, and finally determine which new values to write. Asynchronous automata however have only the last source of non-determinism, since each action determines a fixed subset of processes/registers to read and modify. Deterministic shared-memory systems investigated in [19] have no choice: There is at most one enabled rule for a given action in a given reachable configuration. The expressive power of deterministic shared-memory systems was characterized in [19]. In particular the pomset languages recognized by those systems are regular and consistent, which means that any two prefixes from the language that share a common linear extension must be equal. The reason for that is that each event from this linear order corresponds to a unique rule and the partial order between events derives from the corresponding rules. It is clear that any set of Mazurkiewicz traces (or message sequence charts) is consistent. Now we say that an SMS has *deterministic rules* if the two last sources of non-determinism are void: Formally we require that $(\nu, a, \nu') \in \Delta \wedge (\nu, a, \nu'') \in \Delta$ implies $\nu' = \nu''$. As opposed to determinism, the restriction to shared-memory systems with deterministic rules does not affect their expressive power, as stated by Theorem 2.10 below.

A shared-memory system is *strongly unambiguous* if any pomset admits at most one applicable run. Note here that any deterministic SMS is strongly unambiguous, but not vice-versa. We showed in [20] that a pomset language is recognized by some strongly unambiguous SMS iff it is MSO-definable and *media-bounded*, a new notion similar to being cut-bounded. Consider some $t \in \mathcal{L}$ and some prefix $t' \leqslant t$. An event e in t' is *active* in $\mathcal{L}$ if t' is a prefix of some pomset $t'' \in \mathcal{L}$ in which e is covered by some event $f \in t'' \setminus t'$. A language is called media-bounded if the number of active events in its prefixes is bounded. It is easy to see that any media-bounded language is cut-bounded (Fig. 2). Example 3.1 below will exhibit a pomset language that is cut-bounded but not media-bounded. We consider now another notion of unambiguity closer to the usual ones [2,8]: An SMS $\mathcal{S}$ is simply called *unambiguous* if any pomset from $\mathcal{L}(\mathcal{S})$ admits a unique accepting run. Thus any strongly unambiguous system is unambiguous. Similarly to deterministic rules, this notion of unambiguity does not affect the expressive power of shared-memory systems, as stated by the next result.

Theorem 2.10. *For any shared-memory system $\mathcal{S}$ there exists an unambiguous shared-memory system $\mathcal{S}'$ with deterministic rules such that $\mathcal{L}(\mathcal{S}) = \mathcal{L}(\mathcal{S}')$.*

Proof sketch. Consider a regular and cut-bounded language $\mathcal{L}$. There exists some refinement π from $\mathcal{L}$ onto a regular trace language $\mathcal{L}^\circ \subseteq \mathbb{M}(\Gamma, \|)$ and an asynchronous automaton $\mathcal{A}$ that recognizes $\mathcal{L}^\circ$. We may assume that $\mathcal{A}$ is deterministic. The construction of an SMS $\mathcal{S}$ from $\mathcal{A}$ developped in [20, Lemma 2.6] and already considered above preserves deterministic rules and unambiguity, because π is a bijection from $\mathcal{L}^\circ$ onto $\mathcal{L}$. As a result, $\mathcal{S}$ is unambiguous with deterministic rules. ■

3 Shared-Memory Charts

So far we have established that cut-bounded regular pomset languages are the domain of shared-memory systems, in the same way as Mazurkiewicz traces correspond to asynchronous automata and message sequence charts to message-passing systems. In this section we introduce an algebraic framework for pomset

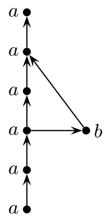


Fig. 3. A flag

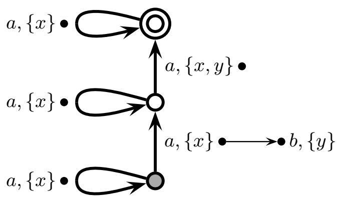


Fig. 4. An SMC specification for flags

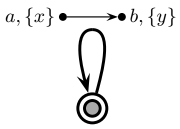


Fig. 5. Ladders

languages in order to give a rational description of regular cut-bounded languages, similarly to analoguous results known for Mazurkiewicz traces and message sequence charts (see [16, Th. 6.4] and [9, Th. 6.3.13]).

EXAMPLE 3.1. Let $\Sigma = \{a, b\}$. A flag is a pomset over Σ that consists of n events labeled by a plus an event labeled by b. It is required that this additional event covers one a-event and is covered by some a-event. An example of flag is depicted in Fig. 3. Since the language of all flags is regular and cut-bounded, it is recognized by some SMS. Note here that this language is not media-bounded. An automaton-based specification for this language is depicted in Fig. 4. The idea is twofold. First each transition of the automaton carries a pomset. Second each event is provided with some additional information, in the abstract form of a subset of gates, that formalizes which events appearing on distinct transitions should be linearly ordered.

3.1 Gates and Shared-Memory Charts

Let G be a finite and non-empty set of *gates*. We put $\Gamma = \Sigma \times 2^G \setminus \{\emptyset\}$. We denote by $\pi_1 : \Gamma \to \Sigma$ and $\pi_2 : \Gamma \to 2^G \setminus \{\emptyset\}$ the two projections and consider the independence relation $\|_\Gamma$ over Γ such that $(a, H)\|_\Gamma(a', H')$ if $H \cap H' = \emptyset$ and $a \neq a'$. Thus two actions from Γ are depend if they carry the same action from Σ or if they share a common gate.

DEFINITION 3.2. *A* shared-memory chart *(for short: an SMC) is a pomset* $t = (E, \preccurlyeq, \xi)$ *over* Γ *such that we have either* $e_1 \preccurlyeq e_2$ *or* $e_2 \preccurlyeq e_1$ *for any two events* e_1 *and* e_2 *with* $\xi(e_1)\not\|_\Gamma\xi(e_2)$*. We denote by* $\mathbb{SMC}$ *the set of all SMCs.*

Thus we require that any two events that carry the same action from Σ or share a common gate must be comparable. It follows that all events sharing a given gate form a linear order. Moreover the structure $\pi_1(t)$ shows no autoconcurrency for any SMC t. Note that shared-memory charts are simply pomsets satisfying the first requirement M_1 of Mazurkiewicz traces over the independence alphabet $(\Gamma, \|_\Gamma)$. However an SMC need not to be a Mazurkiewicz trace over $(\Gamma, \|_\Gamma)$ because we do not require M_2. For that reason the chains defined by gates of an SMC need not to build a chain covering (Def 2.4).

REMARK 3.3. Let $(\Sigma, \|)$ be some finite independence alphabet and $G = \not\| \subseteq \Sigma \times \Sigma$, that is, gates are pairs of dependent actions. Then any Mazurkiewicz trace from $\mathbb{M}(\Sigma, \|)$ can be regarded as an SMC where each event labeled by a

is associated with the set of all gates that contain a. Similarly any message sequence chart can be regarded as an SMC where gates are processes and each event is associated with the (single) process where it occurs. Thus SMCs appear as a generalization of both Mazurkiewicz traces and message sequence charts.

3.2 Asynchronous Product of Shared-Memory Charts

Shared-memory charts are provided with the following concatenation operation. Given two SMCs $t_1 = (E_1, \preccurlyeq_1, \xi_1)$ and $t_2 = (E_2, \preccurlyeq_2, \xi_2)$ the asynchronous product $t_1 \cdot t_2$ is the pomset $t = (E, \preccurlyeq, \xi)$ where $E = E_1 \cup E_2$, $\xi = \xi_1 \cup \xi_2$, and $\preccurlyeq$ is the transitive closure of $\preccurlyeq_1 \cup \preccurlyeq_2 \cup \{(e_1, e_2) \in E_1 \times E_2 \mid \xi(e_1) \not\parallel_\Gamma \xi(e_2)\}$. Thus an event $e_1 \in E_1$ is above an event $e_2 \in E_2$ if they carry the same action from Σ or share a common gate. Clearly the product of two SMCs is again an SMC. This product is associative and admits the empty pomset as unit. It generalizes the product of Mazurkiewicz traces and the product of message sequence charts.

Definition 3.4. *An* SMC specification *is an automaton $\mathcal{A} = (Q, \imath, \longrightarrow, F)$ where Q is a finite set of states, with initial state $\imath$, $\longrightarrow \subseteq Q \times \mathbb{SMC} \times Q$ is a finite set of transitions labeled by SMCs, and $F \subseteq Q$ is a subset of final states.*

As usual, the SMC language $\mathcal{L}_\Gamma(\mathcal{A})$ recognized by an SMC specification $\mathcal{A}$ collects all SMCs obtained as the product of the SMCs along a path from $\imath$ to some final state. On the other hand, we denote by $\mathcal{L}_\Sigma(\mathcal{A})$ the set of pomsets $\mathcal{L}_\Sigma(\mathcal{A}) = \pi_1(\mathcal{L}_\Gamma(\mathcal{A}))$ called *the pomset language recognized by $\mathcal{A}$*. Clearly $\mathcal{L}_\Gamma(\mathcal{A})$ is cut-bounded iff $\mathcal{L}_\Sigma(\mathcal{A})$ is cut-bounded. For convenience we will assume that all states of any SMC specification are reachable from the initial state and co-reachable from the final states.

Remark 3.5. The languages recognized by SMC specifications are precisely the rational languages of the monoid $\mathbb{SMC}$. Similarly to message sequence charts, we could have considered here an equivalent model where SMCs are attached to states instead of transitions.

In the sequel of this section we focus on two natural issues. Given an SMC specification $\mathcal{A}$, we first wonder how to decide whether the pomset language $\mathcal{L}_\Sigma(\mathcal{A})$ is cut-bounded. Second we aim at describing any regular cut-bounded language by some SMC specification.

3.3 Checking Cut-Boundedness of an SMC Specification

Similarly to the characterization of divergence-free MSC specifications [4], we present now a criterion for cut-boundedness of SMC specifications that is based on the following definition of communication graph.

Definition 3.6. *Let $t = (E, \preccurlyeq, \xi)$ be an SMC. The* communication graph *of t is the directed graph $CG(t) = (V, \rightarrow)$ over the set $V = \bigcup_{e \in E} \pi_2(\xi(e))$ of active gates in t such that $g \rightarrow g'$ if*

- *either there are $e, e' \in E$ for which $g \in \pi_2(\xi(e))$, $g' \in \pi_2(\xi(e'))$ and $e \prec\!\cdot\, e'$,*
- *or there are $e, e' \in E$ for which $g \in \pi_2(\xi(e))$, $g' \in \pi_2(\xi(e'))$ and $\xi(e) \not\parallel_\Gamma \xi(e')$.*

Example 3.7. The language of the SMC specification depicted on Fig. 5 consists of all ladders (Fig. 1). The communication graph of any loop is connected but not strongly connected. However, if we add gate x to event b then the language consists of linear orders: It is thus cut-bounded. Moreover the communication graph of *any* loop is strongly connected because of the second condition from Definition 3.6.

Interestingly the communication graph of a message sequence chart [4,16] is very close to the communication graph of the corresponding SMC defined in Remark 3.3: They share a common set of vertices and moreover the connected (resp. strongly connected) components of these two graphs coincide. Now the class of cut-bounded specifications can be characterized similarly to [4, Th. 5].

Theorem 3.8. *The language of an SMC specification is cut-bounded if and only if for any loop* $q_0 \xrightarrow{t_1} q_1 ... q_{n-1} \xrightarrow{t_n} q_n = q_0$*, all connected components of the communication graph* $CG(t_1 \cdot ... \cdot t_n)$ *are strongly connected.*

As a consequence, checking cut-boundedness of a given SMC specification is co-NP. Since this problem coincides with non-divergence in the case of message sequence charts, we know that this problem is actually co-NP-complete [1, Th. 7].

Remark 3.9. Continuing Example 3.7, observe here that the statement of Theorem 3.8 fails if we remove the second condition from Definition 3.6: The simple loop of the SMC specification considered would be connected but not strongly connected although its language is cut-bounded.

3.4 Loop-Connected SMC Specifications

We cannot decide whether an SMC specification describes a regular pomset language, since this question is already undecidable for Mazurkiewicz traces. However, we can exhibit a subclass SMC specifications that describe all regular cut-bounded languages: Similarly to the theory of Mazurkiewicz traces [18,21] or message sequence charts [16], we consider now loop-connected SMC specifications. To do so, we focus on connected SMCs: An SMC is called *connected* if it is connected when regarded as a directed graph on its underlying set of events.

Definition 3.10. *An SMC specification is called* loop-connected *if for all loops* $q_0 \xrightarrow{t_1} q_1 ... q_{n-1} \xrightarrow{t_n} q_n = q_0$ *the SMC* $t_1 \cdot ... \cdot t_n$ *is connected.*

Note that the communication graph of a connected SMC is connected. The converse property fails for shared-memory charts in general, but not for the particular case of message sequence charts. Consequently the notion of a loop-connected SMC specification correspond to globally-cooperative high-level message sequence charts [15] and checking whether an SMC specification is loop-connected is co-NP-complete [15, Prop. 6].

We can prove now the next preliminary observation by means of a folklore technique.

Lemma 3.11. *Let* $\mathcal{L}$ *be a set of* connected *SMCs. If* $\mathcal{L}$ *is MSO-definable then so is* $\mathcal{L}^\star$*, the Kleene iteration of* $\mathcal{L}$ *in the monoid* $\mathbb{SMC}$*.*

It is clear that the SMC language $\mathcal{L}_\Gamma(\mathcal{A})$ of any loop-connected SMC specification $\mathcal{A}$ can be described by a rational expression where iteration operates only over sets of connected SMCs. It follows from Lemma 3.11 that the SMC language $\mathcal{L}_\Gamma(\mathcal{A})$ and the pomset language $\mathcal{L}_\Sigma(\mathcal{A})$ are MSO-definable. If $\mathcal{L}_\Sigma(\mathcal{A})$ is also cut-bounded, then it is regular (Theorem 2.8). Conversely, we can prove that any regular cut-bounded pomset language is described by some loop-connected SMC specification by means of the analoguous result for Mazurkiewicz traces [18,21].

Theorem 3.12. *A cut-bounded language is regular if and only if it is the language of a loop-connected SMC specification.*

Moreover Theorem 3.8 ensures that if a loop-connected SMC specification recognizes some cut-bounded language then the communication graph of any loop is strongly connected. Note that Theorem 3.12 fails if we drop the assumption that the language is cut-bounded: The language of all ladders (Example 2.2) is recognized by a loop-connected SMC specification (Example 3.7) but it is not regular.

4 Related Work

The model of asynchronous cellular automata investigated in [12] can be identified with a subclass of shared-memory systems. However the semantics adopted in [12] differs from ours since the pomset languages recognized by these devices need not to be cut-bounded. Moreover, as opposed to shared-memory systems, asynchronous cellular automata with deterministic rules (called deterministic in [12]) are less expressive than non-deterministic ones, even if one restricts acceptance to some fixed class $\mathbb{P}_k(\Sigma)$.

Concurrency monoids of stably concurrent automata [6,13] form another generalization of Mazurkiewicz traces and message sequence charts which enjoys both a rational and a logical characterization of recognizable languages. These languages are also closely related to *deterministic* shared-memory systems [19]. Consequently this framework cannot deal with non-consistent sets of pomsets such as the language of flags from Example 3.1.

Acknowledgements

Thanks to an anonymous referee of a previous version of this paper for the encoding of Mazurkiewicz traces in shared-memory charts given in Remark 3.3.

References

1. Alur, R., Yannakakis, M.: Model Checking of Message Sequence Charts. In: Baeten, J.C.M., Mauw, S. (eds.) CONCUR 1999. LNCS, vol. 1664, pp. 114–129. Springer, Heidelberg (1999)
2. Arnold, A.: Rational ω-languages are non-ambiguous. Theoretical Computer Science 26, 221–223 (1983)

3. Arnold, A.: An extension of the notion of traces and asynchronous automata. RAIRO, Theoretical Informatics and Applications 25, 355–393 (1991)
4. Ben-Abdallah, H., Leue, S.: Syntactic Detection of Process Divergence and Non-local Choice in Message Sequence Charts. In: Brinksma, E. (ed.) TACAS 1997. LNCS, vol. 1217, pp. 259–274. Springer, Heidelberg (1997)
5. Bernstein, A.J.: Analysis of programs for parallel processing. IEEE Trans. Comp. EC-15(5), 757–762 (1966)
6. Bracho, F., Droste, M., Kuske, D.: Representations of computations in concurrent automata by dependence orders. Theoretical Computer Science 174, 67–96 (1997)
7. Büchi, J.R.: Weak second-order arithmetic and finite automata. Z. Math. Logik Grundlagen Math. 6, 66–92 (1960)
8. Carton, O., Michel, M.: Unambiguous Büchi automata. Theoretical Computer Science 297, 37–81 (2003)
9. Diekert, V., Rozenberg, G.: The Book of Traces. World Scientific, Singapore (1995)
10. Diekert, V., Métivier, Y.: Partial Commutation and Traces. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 457–534 (1997)
11. Dilworth, R.P.: A decomposition theorem for partially ordered sets. Ann. of Math. 51(2), 161–166 (1950)
12. Droste, M., Gastin, P., Kuske, D.: Asynchronous cellular automata for pomsets. Theoretical Computer Science 247, 1–38 (2000)
13. Droste, M., Kuske, D.: Automata with concurrency relations - a survey. In: Advances in Logic, Artificial Intelligence and Robotics, pp. 152–172. IOS Press, Amsterdam (2002)
14. Fanchon, J., Morin, R.: Regular Sets of Pomsets with Autoconcurrency. In: Brim, L., Jančar, P., Křetínský, M., Kučera, A. (eds.) CONCUR 2002. LNCS, vol. 2421, pp. 402–417. Springer, Heidelberg (2002)
15. Genest, B., Muscholl, A., Seidl, H., Zeitoun, M.: Infinite-State High-Level MSCs: Model-Checking and Realizability. J. of Comp. and System Sciences 72, 617–647 (2006)
16. Henriksen, J.G., Mukund, M., Narayan Kumar, K., Sohoni, M., Thiagarajan, P.S.: A Theory of Regular MSC Languages. Information and Computation 202, 1–38 (2005)
17. Kuske, D.: Asynchronous cellular automata and asynchronous automata for pomsets. In: Sangiorgi, D., de Simone, R. (eds.) CONCUR 1998. LNCS, vol. 1466, pp. 517–532. Springer, Heidelberg (1998)
18. Métivier, Y.: On Recognizable Subsets of Free Partially Commutative Monoids. Theoretical Computer Science 58, 201–208 (1988)
19. Morin, R.: Semantics of Deterministic Shared-Memory Systems. In: van Breugel, F., Chechik, M. (eds.) CONCUR 2008. LNCS, vol. 5201, pp. 36–51. Springer, Heidelberg (2008)
20. Morin, R.: Unambiguous Shared-Memory Systems. International Journal of Foundations of Computer Science 21(4), 665–685 (2010)
21. Ochmański, E.: Regular behaviour of concurrent systems. Bulletin of the EATCS 27, 56–67 (1985)
22. Thomas, W.: Languages, automata and logic. In: Rozenberg, G., Salomaa, A. (eds.) Handbook of Formal Languages, vol. 3, pp. 389–455 (1997)
23. Zielonka, W.: Notes on finite asynchronous automata. RAIRO, Theoretical Informatics and Applications 21, 99–135 (1987)

On the CSP Dichotomy Conjecture

Andrei A. Bulatov

School of Computing Science, Simon Fraser University,
Burnaby, Canada
abulatov@cs.sfu.ca

Abstract. We report on the status of the CSP Dichotomy Conjecture and survey recent results and approaches to this problem.

1 Introduction

The Constraint Satisfaction Problem (CSP) has proved to be a very convenient framework that makes it possible to express a wide variety of both practical and theoretical problems. In applications, problems from scheduling, design, computer vision can be uniformly represented as a CSP, and then solved by a universal constraint solver. In theoretical areas, the CSP serves as a natural generalization of graph problems, it is used in logic, and over the last decade its complexity has been a subject of intensive research, including both, the decision and optimization versions.

In this paper we focus on one research problem related to the CSP, or more precisely, its complexity. The Dichotomy Conjecture claims that the CSP parameterized by the set of allowed constraints is either solvable in polynomial time or is NP-complete. Posed by Feder and Vardi in 1993, in spite of significant progress made, it remains open for nearly 20 years. By now it has grown into a beautiful research area that uses and unifies tools from various branches of mathematics and computer science: graph theory, logic, games, and algebra. We outline some of these connections, report on the current status of the dichotomy conjecture, and discuss some ideas on possible future development. We start off with two alternative definitions of the CSP and some important examples. Then we discuss a logic characterization of the CSP, and why this problem is so interesting from the point of view of possible dichotomy results. After that two types of algorithms for the CSP will be explained along with logic formalism, Datalog, capturing one of them. Next we outline the connection between the CSP and algebra, making conjectures and results more precise. In particular, the exact scopes of applicability of the two algorithms will be given. Finally we outline the two existing approaches to combine the two algorithms. Much of the material covered can also be found in [10,11]

2 CSP, Homomorphisms, Logic

The notation we use is fairly standard. The set $\{1,\dots,n\}$ will be denoted by $[n]$. For a set A the set of all n-tuples of elements from A is denoted by A^n. Tuples are denoted in boldface, say, $\mathbf{a}$, and we use $\mathbf{a}[i]$ to refer to the ith entry of tuple $\mathbf{a}$. If φ is a mapping from set A to a set B, we write $\varphi(\mathbf{a})$ to denote the n-tuple $(\varphi(\mathbf{a}[1]),\dots,\varphi(\mathbf{a}[n]))$ from

B^n. An n-ary relation over A is any subset of A^n, and a set of relations over A is called a *constraint language*. Constraint satisfaction problems we consider here are parameterized by constraint languages. Although in most cases a constraint language can be any, finite or infinite, sometimes we will need the assumption that a constraint language is finite.

Definition 1. *Let Γ be a constraint language over a set A. An instance of* $\mathrm{CSP}(\Gamma)$ *is a pair $(V, \mathcal{C})$ where V is a set of* variables, *and $\mathcal{C}$ is a set of* constraints. *Each constraint is again a pair $\langle \mathbf{s}, R\rangle$, where $\mathbf{s}$ is a tuple of variables from V (say, of length k), and R is a k-ary relation from Γ.*

A solution *of $(V, \mathcal{C})$ is given by a mapping $\varphi : V \to A$ such that $\varphi(\mathbf{s}) \in R$ for every constraint $\langle \mathbf{s}, R\rangle \in \mathcal{C}$. The goal in* $\mathrm{CSP}(\Gamma)$ *is, given an instance, decide whether or not it has a solution.*

The following examples of the CSP will frequently occur in the paper.

Example 1. (1) Let $H = (W, E)$ be a (di)graph. In the H-COLORING problem, given a (di)graph G, the goal is to decide whether or not there is an H-coloring of G, that is, an assignment φ of vertices from W to vertices of G such that for every edge vw of G the pair $\varphi(v)\varphi(w)$ is an edge of H. The H-COLORING problem is equivalent to $\mathrm{CSP}(\{E\})$. To see this we treat the set of edges E of H as a binary relation; and convert a given input (di)graph $G = (V, F)$ into a CSP-instance as follows: Let the variables be the vertices of G, and for each $vw \in F$ we introduce a constraint $\langle (v, w), R\rangle$.

(2) In the SATISFIABILITY problem the question is if a given CNF has a satisfying assignment. Every clause in a CNF has the form $v_1^{a_1} \vee \ldots \vee v_k^{a_k}$, where v^a denotes v if $a = 0$ and $\neg v$ if $a = 1$, and satisfied by any assignment of $v_1, \ldots, v_k$ except for $(a_1, \ldots, a_k)$. The corresponding CSP can be defined as follows. For every n and every n-tuple $\mathbf{a}$ from $\{0, 1\}^n$, let $R_{\mathbf{a}} = \{0, 1\}^n - \{\mathbf{a}\}$ be an n-ary relation over $\{0, 1\}$. Then SATISFIABILITY is equivalent to $\mathrm{CSP}(\Gamma_{\mathsf{sat}})$, where the constraint language Γ_{sat} is defined to be $\{R_{\mathbf{a}} \mid \mathbf{a} \in \{0, 1\}^n, n \in \mathbb{N}\}$. Let Φ be a CNF and V its set of variables. The CSP-instance $\mathcal{I}_\Phi$ has the same set V of variables, and for each clause $v_1^{a_1} \vee \ldots \vee v_k^{a_k}$ contains the constraint $\langle (v_1, \ldots, v_k), R_{\mathbf{a}}\rangle$ for $\mathbf{a}$ given by $\mathbf{a}[i] = a_i$, $i \in [k]$.

(3) A linear equation over a finite field naturally represents a constraint on the allowed values of variables it involves. To make the connection more formal assume that we consider equations over $\mathrm{GF}(r)$ and an equation contains k variables. Then the set of assignments satisfying the equation is a coset of a $k-1$-dimensional subspace of $\mathrm{GF}(r)^k$. Let $\Gamma_{\mathsf{lin}(r)}$ denote the set of all relations R over $\mathrm{GF}(r)$ such that R is is a coset of a $k-1$-dimensional subspace of $\mathrm{GF}(r)^k$, and k is the arity of R. It is then straightforward that the problem LINEAR EQUATIONS(r) of checking the consistency of a system of linear equations over $\mathrm{GF}(r)$ is equivalent to $\mathrm{CSP}(\Gamma_{\mathsf{lin}(r)})$.

Feder and Vardi [16] observed that the CSP can also be defined as a problem about homomorphisms. For this alternative definition of the CSP we need a few more definitions. A collection of *relational symbols* $\sigma = \{R_1, \ldots, R_m\}$ each with arity associated with it is called a *vocabulary*. A *relational structure* $\mathcal{A} = (A, R_1^{\mathcal{A}}, \ldots, R_m^{\mathcal{A}})$ with vocabulary σ is a set A enhanced with an r_i-ary relation $R_i^{\mathcal{A}}$ for each R_i. The set A is called the *base set* of $\mathcal{A}$. Sometimes, when it does not lead to a confusion we omit the superscript

$\mathcal{A}$. A *homomorphism* of a relational structure $\mathcal{A} = (A, R_1^{\mathcal{A}}, \dots, R_m^{\mathcal{A}})$ to a relational structure $\mathcal{B} = (B, R_1^{\mathcal{B}}, \dots, R_m^{\mathcal{B}})$ with the same vocabulary is a mapping $\varphi : A \to B$ such that for any R_i and any $\mathbf{a} \in R_i^{\mathcal{A}}$, the image, $\varphi(\mathbf{a})$, belongs to $R_i^{\mathcal{B}}$.

Definition 2. *Let $\mathcal{A}$ be a relational structure over vocabulary σ. In the problem* CSP($\mathcal{A}$) *the question is: Given a relational structure $\mathcal{B}$ over the same vocabulary σ, decide if there is a homomorphism from $\mathcal{B}$ to $\mathcal{A}$.*

As is easily seen for any relational structure $\mathcal{A} = (A, R_1^{\mathcal{A}}, \dots, R_m^{\mathcal{A}})$ the problem CSP$(\mathcal{A})$ is equivalent to the problem CSP$(\Gamma_{\mathcal{A}})$ where $\Gamma_{\mathcal{A}} = \{R_1^{\mathcal{A}}, \dots, R_m^{\mathcal{A}}\}$. Indeed, for any instance $\mathcal{B}$ of CSP$(\mathcal{A})$ we construct an instance $\mathcal{I}_{\mathcal{B}} = (V, \mathcal{C})$ of CSP$(\Gamma_{\mathcal{A}})$ by setting $V = B$ (the base set of $\mathcal{B}$) and including into $\mathcal{C}$ the constraint $\langle \mathbf{a}, R_i^{\mathcal{A}} \rangle$ for each $i \in [m]$ and r_i-tuple $\mathbf{a} \in R_i^{\mathcal{B}}$. For the reverse inversion, given a (finite) constraint language $\Gamma = \{Q_1, \dots, Q_m\}$ on a set A, create a vocabulary $\{R_1, \dots, R_m\}$, set the arity of R_i to be that of Q_i. Then CSP(Γ) is equivalent to CSP$(\mathcal{A}_\Gamma)$ where the base set of $\mathcal{A}_\Gamma$ is A and $R_i^{\mathcal{A}_\Gamma} = Q_i$. Observe that the reverse conversion does not fully work because while a constraint language can be infinite, the set of relational symbols in a vocabulary is finite. Although allowing (formally) relational structures to have infinite vocabularies would fix this problem, some results we mention later depend on the finiteness of a vocabulary.

Example 2. H-COLORING: For (di)graphs H and G every H-coloring of G is a homomorphism from G to H. So the H-COLORING problem is equivalent to CSP(H).

3-SAT: As noted above CSPs requiring infinite constraint languages such as SATISFIABILITY cannot be represented in the homomorphic form. However, restricted versions of SATISFIABILITY can be represented this way, for example, 3-SAT, in which all clauses of a given CNF contains exactly 3 literals. Then 3-SAT is equivalent to CSP$(\mathcal{A}_{\text{3-sat}})$ where $\mathcal{A}_{\text{3-sat}} = (\{0,1\}, R_{\mathbf{a}_1}, \dots, R_{\mathbf{a}_8})$ and $\mathbf{a}_1, \dots, \mathbf{a}_8$ are the eight 3-tuples on $\{0,1\}$. Note that, since only the number of 0s in $\mathbf{a}_i$ matters, the number of relations can be reduced to 4.

LINEAR EQUATIONS(r): By the same reason LINEAR EQUATIONS(r) has to be transformed before converting it into a CSP. Observe that any system of linear equations can be transformed into an equivalent system each equation of which contains at most 3 variables (one may need to add extra variables). Therefore LINEAR EQUATIONS(r) is polynomial time equivalent to CSP$(\mathcal{A}_{\text{3-lin}(r)})$ where the base set of $\mathcal{A}_{\text{3-lin}(r)}$ is GF(r) and the relations are the cosets of 2-dimensional subspaces of GF$(r)^3$.

The homomorphic definition of the CSP allows us to make several simple but useful observations. If there is a homomorphism from relational structure $\mathcal{B}$ to structure $\mathcal{A}$, we write $\mathcal{B} \to \mathcal{A}$. Then CSP$(\mathcal{A})$ will also denote the set $\{\mathcal{B} \mid \mathcal{B} \to \mathcal{A}\}$, that is the set of yes-instances of the problem CSP$(\mathcal{A})$. Relational structures $\mathcal{A}$ and $\mathcal{B}$ are said to be *homomorphically equivalent* if $\mathcal{A} \to \mathcal{B}$ and $\mathcal{B} \to \mathcal{A}$. If $\mathcal{A}, \mathcal{B}$ are homomorphically equivalent, then CSP$(\mathcal{A})$ = CSP$(\mathcal{B})$. A *retraction* of a relational structure $\mathcal{A}$ is a homomorphism $\varphi : \mathcal{A} \to \mathcal{A}$ such that φ acts identically on every element of its image, or, in other words, if $\varphi(\varphi(a)) = \varphi(a)$ for $a \in \mathcal{A}$. A structure is called a *core* if its only retraction is the identity mapping. Any structure $\mathcal{A}$ contains an induced substructure that is a core (take the substructure induced by a minimal image of a retraction), and any two such core substructures are isomorhic. This unique, up to an isomorphism, core substructure is called the *core* of $\mathcal{A}$ and denoted $\mathsf{core}(\mathcal{A})$. As is easily seen, $\mathcal{A}$ and $\mathsf{core}(\mathcal{A})$ are homomorphically equivalent. Therefore in what follows we assume relational structure to be a core every time we need it.

Observe that the class $\mathrm{CSP}(\mathcal{A})$ is closed under homomorphic preimages, that is, if $\mathcal{B} \in \mathrm{CSP}(\mathcal{A})$ and $\mathcal{B}' \to \mathcal{B}$ then $\mathcal{B}' \in \mathrm{CSP}(\mathcal{A})$. With this observation one can easily suggest examples of problems that are not representable in the form $\mathrm{CSP}(\mathcal{A})$ for any $\mathcal{A}$; for example, HAMILTONIAN PATH.

3 CSP vs. MMSNP, Dichotomy

The research direction we focus on in this paper concerns the complexity of the CSP. The early results on the complexity of the CSP characterize problems solvable in polynomial time and suggest that a CSP can be complete in a somewhat limited number of complexity classes. We mention two of these results: The first one is Schaefer's classification of boolean CSPs (or GENERALIZED SATISFIABILITY as he called it), that is, problems $\mathrm{CSP}(\Gamma)$ or $\mathrm{CSP}(\mathcal{A})$ where Γ is a constraint language on $\{0, 1\}$ and $\mathcal{A}$ is a 2-element relational structure [33]. We give details of this result in Section 5. The second one, the complexity classification of the H-COLORING problem by Hell and Nesetril [19]. In both cases every problem considered turned out to be either polynomial time solvable, or NP-complete. Recall, however, that by [28] if P$\neq$NP, there are infinitely many different complexity classes between P and NP.

These two results motivated Feder and Vardi [15,16] to pose a conjecture that is now known as the *Dichotomy Conjecture*.

Conjecture 1 (Dichotomy Conjecture). For any relational structure $\mathcal{A}$ (for any constraint language Γ) the problem $\mathrm{CSP}(\mathcal{A})$ (the problem $\mathrm{CSP}(\Gamma)$) is either polynomial time solvable or NP-complete.

While at that point the Dichotomy Conjecture was a long shot and no particular property that distinguishes the two cases were suggested, later there have been obtained many results that clarifies the boundary between the two cases and witness in favor of the conjecture. We survey many of these results in subsequent sections.

One additional research problem motivated by the Dichotomy Conjecture is the so-called *Meta-Problem*: Given a relational structure $\mathcal{A}$ (a finite constraint language Γ), decide if $\mathrm{CSP}(\mathcal{A})$ is polynomial time solvable. In both cases considered in [33,19] the Meta-Problem is polynomial time solvable.

One of the most intriguing results of [16] suggests that the CSP is in some sense the largest class of problems that can enjoy the dichotomy type properties. Feder and Vardi introduced a logically defined class MMSNP in an attempt to capture the CSP through syntax of logic formulas. Fagin's Theorem [14] states that the class NP consists of problems that can be expressed by existential second-order formulas. In a similar way, the class SNP (Strict NP) [25] consists of problems that can be described by existential second-order formulas with universal first-order part, that is, by formulas $\exists \mathbf{S}' \forall \mathbf{x}\ \Phi(\mathbf{x}, \mathbf{S}, \mathbf{S}')$ where Φ is a quantifier free first-order formula that contains predicates from $\mathbf{S}, \mathbf{S}'$ applied to variables from $\mathbf{x}$. The class of formulas representing CSPs can be narrowed down even more. In a *monadic* SNP formula the predicates from $\mathbf{S}'$ must be unary; in a *monotone* SNP formula the relations from $\mathbf{S}$ have the same *polarity* in Φ; that is, for any $R \in \mathbf{S}$ the polarity of every its occurrence in Φ is either *positive* (if it is contained in an even number of nested subformulas with negation applied to them), or *negative* (if this number is odd). Finally, an SNP formula is said to be *without inequality* if the equality, as well as its negation, inequality, relation is not used in it. The class of problems that can be expressed by an SNP formula that is monotone, monadic, and without inequality is denoted by *MMSNP*.

Theorem 1 ([16,26]). *Every CSP belongs to MMSNP, and every problem from MMSNP is polynomial time equivalent to a CSP.*

That CSP$\subseteq$MMSNP is easily verifiable. Feder and Vardi [16] showed that every problem from MMSNP is equivalent to a CSP up to a polynomial time randomized reduction. Later Kun [26] derandomized this reduction.

If one of these three conditions on the class SNP is omitted, the resulting class of problems is polynomial time equivalent to the entire class NP, and therefore Ladner's result [28] refutes the possibility of a dichotomy for such classes.

Theorem 2 ([16]). *The following classes are polynomial time equivalent to NP: monotone SNP without inequality, monadic SNP without inequality, monotone monadic SNP.*

The class MMSNP admits several alternative characterizations such as *lifts* and *shadows* [27], and *forbidden patterns* [31].

4 Algorithmic Approaches

It would be natural to expect a wide variety of algorithms solving the CSP in those cases in which it can be solved in polynomial time. However, surprisingly, up to now only two types of such algorithms are known, and for each type there is 'the most general' algorithm, which means that basically only two CSP algorithms exist.

4.1 Propagation Algorithms

The first type can be described as *propagation* algorithms. We describe one such algorithm, applicable whenever any other propagation algorithm solves the problem. Let Γ be a constraint language over a set A and $\mathcal{I} = (V, \mathcal{C})$ an instance of CSP(Γ). For an integer $k \geq 1$ the *k-consistency* algorithm [18] works as follows:

- For every k-element set $W \subseteq V$ introduce a new constraint $\langle W, R_W \rangle$ where R_W consists of all *partial solutions* of $\mathcal{I}$, that is, solutions of the problem $\mathcal{I}_W = (W, \mathcal{C}_W)$ such that $\mathcal{C}_W \subseteq \mathcal{C}$ and $\langle \mathbf{s}, R \rangle \in \mathcal{C}$ belongs to $\mathcal{C}_W$ if and only if every entry of $\mathbf{s}$ is in W. Let $\mathcal{I}'$ be the resulting problem.

- Repeat the following step as long as possible: If there is a k-element set $W \subseteq V$, a variable $v \in V - W$, and a tuple $\mathbf{a} \in R_W$ such that $\mathbf{a}$ cannot be extended to a solution of $\mathcal{I}'_{W \cup \{v\}}$, remove $\mathbf{a}$ from R_W and update $\mathcal{I}'$.

It is not hard to see [18] that if the k-consistency returns an instance with empty relations R_W then $\mathcal{I}$ has no solution. The converse is however not true, see Example 3(4) below. We say that the k-consistency algorithms solves $\mathrm{CSP}(\Gamma)$ if every instance $\mathcal{I}$ of $\mathrm{CSP}(\Gamma)$, on which the algorithm returns non-empty constraints, has a solution. The minimal k such that the k-consistency algorithm solves $\mathrm{CSP}(\Gamma)$ is called the *width* of the problem. The problem $\mathrm{CSP}(\Gamma)$ is said to be of *bounded width* if it has width k for some k.

Example 3. (1) The 2-SAT problem has bounded width, more precisely, width 2.

(2) The H-COLORING problem has width 2 when graph H is bipartite, and NP-complete otherwise.

(3) The HORN-SAT is the SATISFIABILITY problem restricted to Horn clauses, i.e. clauses of the form $x_1 \wedge \ldots \wedge x_k \rightarrow y$. Let $\Gamma_{\text{k-Horn}}$ be the constraint language consisting of relations expressible by a Horn clause with at most k premises. The problem k-HORN-SAT equivalent to $\mathrm{CSP}(\Gamma_{\text{k-Horn}})$ and allowing only clauses of restricted arity has width k.

(4) LINEAR EQUATIONS provides an archetypical example of a CSP that does not have bounded width. Indeed, for any k consider the system of equations given by $x_{21} + x_{12} + x_{k+21} + x_{1k+2} = 1$ and $x_{i+1j} + x_{ij+1} + x_{i-1j} + x_{ij-1} = 0$ for any other $i, j \in [k+2]$. While this system is inconsistent, the k-consistency algorithm returns an instance with non-empty constraints.

4.2 Gaussian Elimination

The simplest algorithm of the second type is known from basic linear algebra — Gaussian elimination. While propagation algorithms cannot solve LINEAR EQUATIONS, it is solvable by Gaussian elimination. A similar algorithm solving *group constraints*, defined in terms of finite groups, was suggested in [16]. In Section 6 we consider further generalizations of the Gaussian elimination algorithm.

4.3 Logic Characterizations

The two types of algorithms were first identified in [16]. In the same paper Feder and Vardi suggested a logic framework for problems of bounded width — Datalog. *Datalog* is a language of logic programming that can be applied to relational structures of suitable vocabulary. It uses two types of predicate symbols. Symbols of the first type, *Extensional Database Predicates* or *EDB* are the relations of the input structure and do not change in the course of the execution of the program. Symbols of the second type, *Intensional Database Predicates* or *IDB*, do not belong to the vocabulary of the input structure and change until a fixed point is reached. The semantics of Datalog is best illustrated by the following example program consisting of three *rules* that works on graphs:

$$\begin{aligned}
&\mathit{oddpath}(x,y) &&:- \ \mathit{edge}(x,y)\\
&\mathit{oddpath}(x,y) &&:- \ \mathit{oddpath}(x,z), \mathit{edge}(z,t), \mathit{edge}(t,y)\\
&\mathit{oddcycle} &&:- \ \mathit{oddpath}(x,x)
\end{aligned}$$

The predicate *edge* is an EDB and *edge*(v, w) indicates that there is an edge between v and w in the input graph. The first rule states that IDB *oddpath*(v, w) is true whenever vw is an edge. Then the second rule amounts to say that whenever *oddpath*(v, w) is true and there is a path of length two from w to u, then *oddpath*(v, u) is also true. Thus, *oddpath* has to be true on all pairs of vertices connected with a path of odd length. Finally, the second IDB, *oddcycle* is true if and only if the input graph contains an odd cycle. In this sense the Datalog program decides whether or not a graph is bipartite, or, equivalently, solves the problem $\mathrm{CSP}(H)$, where H is a bipartite graph.

More precisely, let $\mathcal{A}$ be a relational structure and P a Datalog program with a designated null-ary IDB Q. Then P solves the problem $\mathrm{CSP}(\mathcal{A})$ if, for any structure $\mathcal{B}$ with the same vocabulary as $\mathcal{A}$, the program P applied to $\mathcal{B}$ ends up with Q true if and only if $\mathcal{B} \not\in \mathrm{CSP}(\mathcal{A})$. The left side of a Datalog rule is called its *head*, and the right side of the rule is called the *body*.

Theorem 3 ([24]). *For a relational structure $\mathcal{A}$ the problem $\mathrm{CSP}(\mathcal{A})$ has width k if and only if there is a Datalog program P that solves $\mathrm{CSP}(\mathcal{A})$ such that each rule of P contains at most $k + 1$ variable, and the head of each rule contains at most k variable.*

In an attempt to characterize relational structures that give rise to problems of bounded width [16] introduces the property of ability to count that in a nutshell means that LINEAR EQUATIONS(2) can be simulated via $\mathrm{CSP}(\mathcal{A})$. It proves that if a structure $\mathcal{A}$ has the ability to count then $\mathrm{CSP}(\mathcal{A})$ does not have bounded width, and conjectures that the reverse is also true.

Gaussian elimination type algorithms turned out to be more difficult to formalize. So far no logic condition is known that would capture, say, LINEAR EQUATIONS(2). On the contrary, [1] shows that this problem eludes even very powerful logic languages.

Unfortunately, logic characterizations of the CSP and problems of bounded width do not allow one to formulate a precise criterion distinguishing polynomial time solvable CSPs from NP-complete ones, nor to give a concise and verifiable characterization of problems of bounded width.

5 Algebraic Background

5.1 Polymorphisms

In [23] Jeavons et al. observed that certain invariance properties of constraints completely characterize the complexity of the CSP. Let R be a (say, n-ary) relation over a set A. An operation $f : A^m \to A$ is called an *polymorphism* of R if for any $\mathbf{a}_1, \dots, \mathbf{a}_m \in R$ the tuple $f(\mathbf{a}_1, \dots, \mathbf{a}_m)$ obtained by component-wise application of f also belongs to R. Operation f is a polymorphism of a constraint language Γ over A (or a relational structure $\mathcal{A} = (A, R_1, \dots, R_k)$) if it is a polymorphism of each relation from Γ (each relation R_i). The set of all polymorphisms of Γ and $\mathcal{A}$ is denoted by $\mathsf{Pol}(\Gamma)$, $\mathsf{Pol}(\mathcal{A})$, respectively. For a set F of operations on A by $\mathsf{Inv}(F)$ we denote the set of relations on A that are *invariant* under each $f \in F$, that is, f is a polymorphism of all relations in $\mathsf{Inv}(F)$.

Theorem 4 **([23]).** *Let Γ_1, Γ_2 be constraint languages over the same set. If* $\mathsf{Pol}(\Gamma_1) \subseteq \mathsf{Pol}(\Gamma_2)$ *then* $\mathrm{CSP}(\Gamma_2)$ *is polynomial time reducible to* $\mathrm{CSP}(\Gamma_1)$[1].

Theorem 4 made it possible to state some of the existing complexity results in a more concise form, and also to discover new more general classes of CSPs solvable in polynomial time or having bounded width. Some of these results are summarized in the following example.

Example 4 ([23,22,8]). (1) A binary operation f on a set A is said to be *semilattice* if for any $x, y \in A$ the following equations hold: $f(x,x) = x$, $f(x,y) = f(y,x)$, $f(f(x,y),z) = f(x,f(y,z))$. If a relational structure $\mathcal{A}$ has a semilattice polymorphism then $\mathrm{CSP}(\mathcal{A})$ has bounded width. (Note that this property may not be true for infinite constraint languages. Although in this case a different notion of width works, see, e.g. [12].)

(2) A k-ary operation g on A is called a *near-unanimity* operation, or *NU* if $g(y,x,\dots,x) = g(x,y,x,\dots,x) = \dots = g(x,\dots,x,y) = x$ for any $x, y \in A$. A ternary NU is also referred to as a *majority* operation. If a relational structure $\mathcal{A}$ has an NU polymorphism then $\mathrm{CSP}(\mathcal{A})$ has bounded width.

(3) A ternary operation h on A is called *Mal'tsev* if $h(x,y,y) = h(y,y,x) = x$ for any $x, y \in A$. If a relational structure $\mathcal{A}$ has a Mal'tsev polymorphism then $\mathrm{CSP}(\mathcal{A})$ is solvable in polynomial time, although does not necessarily have bounded width. The structure $\mathcal{A}_{\mathsf{3\text{-}lin}(r)}$ encoding LINEAR EQUATIONS(r) has the Mal'tsev polymorphism $x - y + z$ where $+$ and $-$ are the operations of $\mathrm{GF}(r)$.

(4) A structure $\mathcal{A}$ is a core if and only if every its unary polymorphism is a bijective mapping. If $\mathcal{A}$ is a core and every its polymorphism f is such that $f(x_1,\dots,x_n) = x_i$ for some i and all $x_1,\dots,x_n \in \mathcal{A}$ then $\mathrm{CSP}(\mathcal{A})$ is NP-complete.

(5) Schaefer's Theorem [33] can be stated in terms of polymorphisms. Let $\mathcal{A}$ be a 2-element relational structure (we assume its base set to be $\{0,1\}$) which is a core. The problem $\mathrm{CSP}(\mathcal{A})$ is solvable in polynomial time if and only if one of the following operations is a polymorphism of $\mathcal{A}$: semilattice operations of conjunction and disjunction, the majority operation on $\{0,1\}$ (there is only one such operation), or a Mal'tsev operation $x - y + z$ where $+$ and $-$ are modulo 2. Otherwise $\mathrm{CSP}(\mathcal{A})$ is NP-complete.

5.2 Algebras

A *(universal) algebra* is a set equipped with a collection of operations on it. It is convenient to represent an algebra as a pair $\mathbb{A} = (A, F)$ where A is the *base set* or the *universe* of $\mathbb{A}$ and F is the set (finite or infinite) of its *basic operations*; operations from $\mathsf{Pol}(\mathsf{Inv}(F))$ are called *term* operations of $\mathbb{A}$. Any relational structure $\mathcal{A}$ (with base set A) can be paired up with the algebra $\mathsf{Alg}(\mathcal{A}) = (A, \mathsf{Pol}(\mathcal{A}))$. On the other hand, an algebra $\mathbb{A} = (A, F)$, can be associated with a class of problems $\mathrm{CSP}(\mathcal{A})$ where $\mathcal{A} = (A, R_1, \dots, R_k)$ is such that $R_1, \dots, R_k \in \mathsf{Inv}(F)$, denoted by $\mathrm{CSP}(\mathbb{A})$.

[1] Using the result of [32] that the ST-CONNECTIVITY problem is in L the reduction can be made log space. Later Larose and Tesson [29] proved that it can even be made AC_0.

If all problems from $\mathrm{CSP}(\mathbb{A})$ are solvable in polynomial time $\mathbb{A}$ is called *tractable*, if $\mathrm{CSP}(\mathbb{A})$ contains an NP-complete problem, $\mathbb{A}$ is called *NP-complete*.

An algebra $\mathbb{A} = (A, F)$ is said to be *idempotent* if $f(x, \dots, x) = x$ for any $f \in F$ and any $x \in A$. Any algebra can be made idempotent, $\mathsf{Id}(\mathbb{A})$ denotes the *idempotent reduct* of $\mathbb{A}$, the algebra (A, F') where F' is the set of all idempotent term operations of $\mathbb{A}$. If a relational structure $\mathcal{A}$ is a core then $\mathsf{Alg}(\mathcal{A})$ is tractable (NP-complete) if and only if $\mathsf{Id}(\mathsf{Alg}(\mathcal{A}))$ is tractable (NP-complete) [9]. Thus, we may assume all algebras we deal with to be idempotent.

Relations from $\mathsf{Inv}(F)$ can also be viewed in the algebraic way. The *kth power* of $\mathbb{A}$ is the algebra $\mathbb{A}^k$ such that the universe of $\mathbb{A}^k$ is A^k, and for each basic operation $f \in F$ algebra $\mathbb{A}^k$ has the basic operation f^k that acts on elements of A^k, that is, k-tuples over A, component-wise as f. If $B \subseteq A$ is such that $f(x_1, \dots, x_n) \in B$ for any $f \in F$ and $x_1, \dots, x_n \in B$ the algebra $\mathbb{B} = (B, F_B)$, where operations in F_B are restrictions of those from F onto B, is called a *subalgebra* of $\mathbb{A}$. As is easily seen, a k-ary relation from $\mathsf{Inv}(F)$ is a subalgebra of $\mathbb{A}^k$.

In the 1980's Hobby and McKenzie developed tame congruence theory that studies the local structure of algebras [20]. They discovered that the local structure of universal algebras is surprisingly well behaved and can be classified into just five types. Each type is associated with a certain basic algebra, and if an algebra *admits* a type, it means that its local structure resembles that of the corresponding basic algebra. The five basic algebras and corresponding types are:

1. A unary algebra whose basic operations are all permutations (unary type);
2. A one-dimensional vector space over some finite field (affine type);
3. A 2-element *boolean algebra* whose basic operations include conjunction, disjunction, and negation (boolean type);
4. A 2-element *lattice* whose basic operations include conjunction and disjunction (lattice type);
5. A 2-element *semilattice* whose basic operations include a semilattice operation (semilattice type).

Omitting or admitting types is strongly related to the complexity of the CSP.

Theorem 5 ([9]). *If a relational structure $\mathcal{A}$ is such that $\mathsf{Alg}(\mathcal{A})$ is idempotent and admits the unary type then $\mathrm{CSP}(\mathcal{A})$ is NP-complete.*

Theorem 5 allows one to make the Dichotomy Conjecture more precise.

Conjecture 2. If a relational structure $\mathcal{A}$ is such that $\mathsf{Alg}(\mathcal{A})$ is idempotent, then $\mathrm{CSP}(\mathcal{A})$ is solvable in polynomial time if and only if $\mathsf{Alg}(\mathcal{A})$ does not admit the unary type. Otherwise it is NP-complete.

If Conjecture 2 is true, the Meta-Problem (that is, given a relational structure $\mathcal{A}$ or an algebra $\mathbb{A}$, deciding the complexity of $\mathrm{CSP}(\mathcal{A})$, $\mathrm{CSP}(\mathbb{A})$) is solvable in polynomial time if an algebra is given, or the size of the base set of a relational structure is bounded [17]. In the remaining case of an arbitrary relational structure its complexity is unknown.

6 Algebras and Algorithms

6.1 Datalog and Bounded Width

The property of a relational structure to have ability to count can be expressed in algebraic terms. Larose et al. [30] proved that a relational structure $\mathcal{A}$ does not have the ability to count (and therefore can have bounded width) if and only if $\mathsf{Alg}(\mathcal{A})$ omits the unary and affine types. Finally, Barto and Kozik, and Bulatov independently proved this algebraic characterization of bounded width.

Theorem 6 ([3,7]). *For a relational structure $\mathcal{A}$ the problem* $\mathrm{CSP}(\mathcal{A})$ *has bounded width if and only if $\mathsf{Alg}(\mathcal{A})$ omits the unary and affine types.*

The Meta-Problem in this case is solvable in polynomial time.

6.2 Gaussian Elimination and Few Subpowers

Algebraic techniques make it possible to generalize the Gaussian elimination algorithm. The algorithm from [8] solving $\mathrm{CSP}(\mathcal{A})$ for a relational structure $\mathcal{A}$ with a Mal'tsev polymorphism can be viewed as a generalization of Gaussian elimination in the following sense. Similar to the output of Gaussian elimination it constructs some sort of a basis or a *compact representation* of the set of all solutions of a CSP. To generate such a compact representation the algorithm uses the property of rectangularity of relations with a Mal'tsev polymorphism. A (say, n-ary) relation R over a set A is said to be *rectangular* if for any $\mathbf{a}, \mathbf{b}, \mathbf{c} \in R$ such that $\mathbf{a}[i] = \mathbf{b}[i]$ for $i \in [n-1]$ and $\mathbf{b}[n] = \mathbf{c}[n]$ the tuple $\mathbf{d}$ given by $\mathbf{d}[i] = \mathbf{c}[i]$ for $i \in [n-1]$ and $\mathbf{d}[n] = \mathbf{a}[n]$ also belongs to R.

Due to rectangularity if R has a Mal'tsev polymorphism, it admits a simple description. Let $\mathrm{Sig}(R)$ be a set of triples $(i, a, b), i \in [n], a, b \in A$, given by: $(i, a, b) \in \mathrm{Sig}(R)$ if and only if there are tuples $\mathbf{a}, \mathbf{b} \in R$ such that $\mathbf{a}[j] = \mathbf{b}[j]$ for $j \in [i-1]$, and $\mathbf{a}[i] = a$, $\mathbf{b}[i] = b$. Such a pair of tuples is said to *witness* (i, a, b). Rectangularity implies that if $(i, a, b) \in \mathrm{Sig}(R)$ then for any $\mathbf{a} \in R$ with $\mathbf{a}[i] = a$ there is $\mathbf{b} \in R$ such that $\mathbf{a}, \mathbf{b}$ witness (i, a, b). A compact representation of R is any set of tuples Q from R such that for any $(i, a, b) \in \mathrm{Sig}(R)$ it contains a pair of tuples witnessing the triple.

The relation R can be reconstructed from its compact representation Q by means of a Mal'tsev polymorphism h. Indeed, suppose $\mathbf{a} \in R$ and we have managed to reconstruct a tuple $\mathbf{b}$ such that $\mathbf{b}[j] = \mathbf{a}[j]$ for $j \in [i-1]$. Then $\mathbf{a}, \mathbf{b}$ witness that $(i, \mathbf{a}[i], \mathbf{b}[i]) \in \mathrm{Sig}(R)$, and therefore there are $\mathbf{c}, \mathbf{d} \in Q$ that witness this. Using Mal'tsev identities we see that the tuple $\mathbf{e} = h(\mathbf{a}, \mathbf{c}, \mathbf{d})$ is such that $\mathbf{e}[j] = \mathbf{a}[j]$ for $j \in [i]$.

Suppose $\varGamma$ is a constraint language over A with a Mal'tsev polymorphism. Given an instance $(V, \mathcal{C})$ of $\mathrm{CSP}(\varGamma)$ the algorithm considers the set of all solutions as a single relation. It starts with a compact representation of $R_0 = A^{|V|}$, and then adds constraints from $\mathcal{C}$ one by one, generating relations $R_1, \ldots, R_{|\mathcal{C}|}$ such that R_i consists of the tuples (assignments to variables from V) that satisfy the first i constraints. Every relation R_i is given by its compact representation, which is used to compute a compact representation of R_{i+1}.

Dalmau [13] generalized this algorithm so that it is applicable to constraint languages with a *generalized majority minority* (*GMM*) polymorphism. A GMM operation sometimes behaves like a Mal'tsev operation, and sometimes as an NU operation.

It is thought that the property of relations to have a compact representation, where compactness is understood as having size polynomial in the arity of the relation, is the right generalization of linear algebra problems where Gaussian elimination can be used. Let $\mathbb{A} = (A, F)$ be an algebra. It is said to be an algebra *with few subpowers* if every relation over A invariant under F admits a compact representation [21]. The term 'few subpowers' comes from the observation that every relation invariant under F is a subalgebra of a direct power of $\mathbb{A}$, and if the size of compact representation is bounded by a polynomial $p(n)$ then at most $2^{p(n)}$ n-ary relations can be represented, while the total number of such relations can be as large as $2^{|A|^n}$.

Algebras with few subpowers are completely characterized by Idziak et al. [21].

Theorem 7 ([21]). *An algebra $\mathbb{A}$ is an algebra with few subpowers if and only if for some k it has a k-edge term operation f, that is a $k+1$-ary operation satisfying for any $x, y \in \mathbb{A}$ the following identities:*

$$\begin{aligned}
f(x, x, y, y, y, \ldots, y, y) &= y, \\
f(x, y, x, y, y, \ldots, y, y) &= y, \\
f(y, y, y, x, y, \ldots, y, y) &= y, \\
f(y, y, y, y, x, \ldots, y, y) &= y, \\
&\cdots \\
f(y, y, y, y, y, \ldots, y, x) &= y.
\end{aligned}$$

A minor generalization of the algorithm from [13] solves $\mathrm{CSP}(\Gamma)$ in which Γ consists of relations admitting compact representation.

The complexity of the Meta-Problem for algebras with few subpowers (deciding whether or no a given relational structure gives rise to an algebra with few subpowers) is unknown.

7 Combining the Two Approaches

As we discussed in Section 6, the two algorithmic approaches to the CSP are well understood. However, for a more general CSP one is likely to need a combination of them. In this section we consider some ideas and results on possible combined algorithms.

The two general dichotomy results that show how the approaches interact are the dichotomy theorem for 3-element relational structures [6] and that for conservative structures [4], see also [2]. Recall that a relational structure $\mathcal{A}$ is said to be *conservative* if for any its polymorphism f (say, n-ary) and any $x_1, \ldots, x_n \in \mathcal{A}$ we have $f(x_1, \ldots, x_n) \in \{x_1, \ldots, x_n\}$. Conservative structures correspond to so-called *List Homomorphism* problems, where one can specify for each variable a set of its allowed values. In both cases Conjecture 2 is confirmed.

We give some details on the approach taken in [4,2] and then discuss how it can be generalized. We proceed in several steps. First, given a relational structure $\mathcal{A}$ we show how to isolate the parts of $\mathcal{A}$ that look similar to those having few subpowers, and therefore problems over such parts can hopefully be solved by a GMM-type algorithm. Then, given an instance $\mathcal{I}$ of $\mathrm{CSP}(\mathcal{A})$ we show how after applying a propagation algorithm $\mathcal{I}$

can be split into subproblems that satisfy three conditions: (1) Each of the subproblems is a problem over some 'few subpowers' parts, (2) if all the subproblems have a solution then $\mathcal{I}$ has a solution, and (3) if one of the subproblems does not have a solution, $\mathcal{I}$ can be simplified by reducing the sets of possible values of some variables.

Since $\mathcal{A}$ is conservative, every its 2-element subset B is a subalgebra. As we assume that $\mathrm{CSP}(\mathcal{A})$ is solvable in polynomial time, there must be a polymorphism that on B is one of the operations listed in Schaefer's Theorem (Example 4(5)). More precisely, there are polymorphisms f, g, h of $\mathcal{A}$ such that all pairs ab of elements of $\mathcal{A}$ (we call them *edges*) can be divided up into three categories: *semilattice* edges, on which f is a semilattice operation; *majority* edges, which are not semilattice, but g is a majority operation on each of them; *affine* edges, which are not of the previous types, and h is a Mal'tsev operation on each of them. Then $\mathcal{A}$ is treated as a complete graph whose edges are colored with the three colors. Note that semilattice edges are directed since elements are not symmetrical with respect to a semilattice operation. Now the 'few subpowers' parts of $\mathcal{A}$ are defined to be the minimal subsets $D \subseteq \mathcal{A}$ such that there are no semilattice or affine edge ab with $a \in D$, $b \notin D$. Such subsets are called *as-components*.

In [2] the role of as-components is played by minimal absorbing subuniverses. Let f be an n-ary term operation of $\mathbb{A} = \mathsf{Alg}(\mathcal{A})$. A set $D \subseteq \mathcal{A}$ is called an *absorbing subuniverse with respect to* f, if D is a subalgebra of $\mathbb{A}$ and for any $k \in [n]$ and any $x_1, \dots, x_n \in \mathcal{A}$ such that $x_i \in D$ for all $i \neq k$, the element $f(x_1, \dots, x_n)$ belongs to D. If D is a proper subset of $\mathcal{A}$, the absorbing subuniverse is called *proper*. The absorbing subuniverse D is called *minimal* if subalgebra $\mathbb{D} = (D, \mathsf{Pol}(\mathcal{A})_D)$ does not have proper absorbing subuniverses with respect to any its term operation.

Let $\mathcal{I} = (V, \mathcal{C})$ be an instance of $\mathrm{CSP}(\mathcal{B})$, and the 2-consistency algorithm being applied to $\mathcal{I}$ does not change it. Let S_v, S_{vw} denote the set of possible values of v, and the set of pairs of values that can be taken by variables $v, w \in V$ simultaneously. It can be proved that there are as-components $D_v \subseteq S_v$ for each $v \in V$ such that $S_{vw} \cap (D_v \times D_w) \neq \emptyset$ for any $v, w \in V$. Two variables are called *connected* if for any $(a, b) \in S_{vw}$ we have $a \in D_v$ if and only if $b \in D_w$. The connectedness of variables defines a partition of V into classes $V_1, \dots, V_k$. The problem $\mathcal{I}$ can now be split into subproblems $\mathcal{I}_1, \dots, \mathcal{I}_k$ as follows: Set $\mathcal{I}_j = (V_j, \mathcal{C}_j)$, where for every $C = \langle \mathbf{s}, R \rangle$, $\mathbf{s} = (v_1, \dots, v_n)$, the set $\mathcal{C}_j$ includes the constraint $C_j = \langle \mathbf{s}_j, R_j \rangle$ such that $\mathbf{s}_j = (v_{i_1}, \dots, v_{i_\ell})$ consists of all elements of $\mathbf{s}$ belonging to V_j, and $R_j = \{(\mathbf{a}[i_1], \dots, \mathbf{a}[i_\ell]) \mid \mathbf{a} \in R\}$.

To solve the problems $\mathcal{I}_j$ it is either possible to use the GMM algorithm straightforwardly, or $\mathcal{I}_j$ can be further reduced using a different approach and then processed by the GMM algorithm. If one of the $\mathcal{I}_j$ has no solution then the problem $\mathcal{I}$ can be simplified by removing as-components D_v, $v \in V_j$, from sets S_v and starting the procedure again. If all the $\mathcal{I}_j$ have a solution, they combine to produce a solution of $\mathcal{I}$.

Proposition 1. *If every $\mathcal{I}_j$ has a solution φ_j, $\mathcal{I}$ has a solution φ such that $\varphi(v) = \varphi_j(v)$ whenever $v \in V_j$.*

Switching to minimal absorbing subuniverses does not change the overall approach, only solving the problems $\mathcal{I}_j$ is significantly simpler.

Some of these steps remain working in the case of general relational structures. Absorbing subuniverses do not depend whether or not the structure is conservative. It is shown in [5] that an edge-colored graph can be defined for arbitrary relational structure as well. Splitting a problem can also be done in a similar way. However, if some $\mathcal{I}_j$ has no solution, the problem cannot be simplified in the same straightforward way. Also Proposition 1 is not true anymore. We believe that overcoming these two difficulties either using colored graphs or absorbing subuniverses will eventually lead to settling the Dichotomy Conjecture.

References

1. Atserias, A., Bulatov, A., Dawar, A.: Affine systems of equations and counting infinitary logic. In: ICALP, pp. 558–570 (2007)
2. Barto, L.: The dichotomy for conservative constraint satisfaction problems revisited (2011)
3. Barto, L., Kozik, M.: Constraint satisfaction problems of bounded width. In: FOCS, pp. 595–603 (2009)
4. Bulatov, A.: Tractable conservative constraint satisfaction problems. In: Proceedings of the 18th Annual IEEE Simposium on Logic in Computer Science, Ottawa, Canada, pp. 321–330. IEEE Computer Society, Los Alamitos (2003)
5. Bulatov, A.: A graph of a relational structure and constraint satisfaction problems. LICS, pp. 448–457 (2004)
6. Bulatov, A.: A dichotomy theorem for constraint satisfaction problems on a 3-element set. J. ACM 53(1), 66–120 (2006)
7. Bulatov, A.: Bounded relational width (2009)
8. Bulatov, A., Dalmau, V.: A simple algorithm for Mal'tsev constraints. SIAM J. Comput. 36(1), 16–27 (2006)
9. Bulatov, A., Jeavons, P., Krokhin, A.: Classifying the complexity of constraints using finite algebras. SIAM J. Comput. 34(3), 720–742 (2005)
10. Bulatov, A., Krokhin, A., Larose, B.: Dualities for constraint satisfaction problems. In: Complexity of Constraints, pp. 93–124 (2008)
11. Bulatov, A., Valeriote, M.: Recent results on the algebraic approach to the CSP. In: Complexity of Constraints, pp. 68–92 (2008)
12. Dalmau, V., Pearson, J.: Set functions and width 1 problems. In: Jaffar, J. (ed.) CP 1999. LNCS, vol. 1713, pp. 159–173. Springer, Heidelberg (1999)
13. Dalmau, V.: Generalized majority-minority operations are tractable. Logical Methods in Computer Science 11(1) (2005) (electronic)
14. Fagin, R.: Generalized first order spectra, and polynomial time recognizable sets. In: Complexity of Computations (1974)
15. Feder, T., Vardi, M.Y.: Monotone monadic SNP and constraint satisfaction. In: Proceedings of 25th ACM Symposium on the Theory of Computing (STOC), pp. 612–622 (1993)
16. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. SIAM Journal of Computing 28, 57–104 (1998)
17. Freese, R., Valeriote, M.: On the complexity of some maltsev conditions. IJAC 19(1), 41–77 (2009)
18. Freuder, E.C.: Synthesizing constraint expressions. Communications of the ACM 21, 958–966 (1978)
19. Hell, P., Nešetřil, J.: On the complexity of H-coloring. Journal of Combinatorial Theory, Ser.B 48, 92–110 (1990)

20. Hobby, D., McKenzie, R.N.: The Structure of Finite Algebras. Contemporary Mathematics, vol. 76. American Mathematical Society, Providence (1988)
21. Idziak, P., Markovic, P., McKenzie, R., Valeriote, M., Willard, R.: Tractability and learnability arising from algebras with few subpowers. SIAM J. Comput. 39(7), 3023–3037 (2010)
22. Jeavons, P., Cohen, D., Cooper, M.: Constraints, consistency and closure. Artificial Intelligence 101(1-2), 251–265 (1998)
23. Jeavons, P., Cohen, D., Gyssens, M.: Closure properties of constraints. Journal of the ACM 44, 527–548 (1997)
24. Kolaitis, P., Vardi, M.: A game-theoretic approach to constraint satisfaction. In: Proceedings of the 17th National (US) Conference on Artificial Intelligence, AAAI 2000, pp. 175–181 (2000)
25. Kolaitis, P., Vardi, M.: The decision problem for the probabilities of higher-order properties. In: STOC, pp. 425–435 (1987)
26. Kun, G.: Constraints, MMSNP and expander relational structures. Mathematics, abs/0706.1701 (2007)
27. Kun, G., Nešetřil, J.: NP by means of lifts and shadows. In: Kučera, L., Kučera, A. (eds.) MFCS 2007. LNCS, vol. 4708, pp. 171–181. Springer, Heidelberg (2007)
28. Ladner, R.: On the structure of polynomial time reducibility. Journal of the ACM 22, 155–171 (1975)
29. Larose, B., Tesson, P.: Universal algebra and hardness results for constraint satisfaction problems. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 267–278. Springer, Heidelberg (2007)
30. Larose, B., Valeriote, M., Zádori, L.: Omitting types, bounded width and the ability to count. IJAC 19(5), 647–668 (2009)
31. Madelaine, F., Stewart, I.: Constraint satisfaction, logic and forbidden patterns. SIAM J. Comput. 37(1), 132–163 (2007)
32. Reingold, O.: Undirected st-connectivity in log-space. In: STOC, pp. 376–385 (2005)
33. Schaefer, T.: The complexity of satisfiability problems. In: Proceedings of the 10th ACM Symposium on Theory of Computing (STOC 1978), pp. 216–226 (1978)

LR(0) Conjunctive Grammars and Deterministic Synchronized Alternating Pushdown Automata

Tamar Aizikowitz and Michael Kaminski

Department of Computer Science,
Technion – Israel Institute of Technology,
Haifa 32000, Israel

Abstract. In this paper we introduce a sub-family of synchronized alternating pushdown automata, *Deterministic Synchronized Alternating Pushdown Automata*, and a sub-family of conjunctive grammars, $LR(0)$ *Conjunctive Grammars*. We prove that deterministic SAPDA and $LR(0)$ conjunctive grammars have the same recognition/generation power, analogously to the classical equivalence between acceptance by empty stack of deterministic PDA and $LR(0)$ grammars. These models form the theoretical basis for efficient, *linear*, parsing of a rich sub-family of conjunctive languages, which *properly* includes all the boolean combinations of context-free $LR(0)$ languages.

1 Introduction

Context-free languages lay at the very foundations of Computer Science, proving to be one of the most appealing language classes for practical applications. On the one hand, they are quite expressive, covering such syntactic constructs as necessary, e.g., for mathematical expressions. On the other hand, they are polynomially parsable, making them practical for real world applications. However, research in certain fields, e.g., Computational Linguistics [4,8], has raised a need for computational models which extend context-free languages.

Conjunctive Grammars (CG) are an example of such a model. Introduced by Okhotin in [9], CG are a generalization of context-free grammars which allow explicit conjunction operations in rules, thereby adding the power of intersection. CG were shown by Okhotin to accept all finite intersections of context-free languages, as well as some additional languages. Okhotin proved the languages generated by these grammars to be polynomially parsable [9,12,13], making the model practical from a computational standpoint, and therefore, of interest for applications in various fields such as, e.g., programming languages.

Alternating automata models were first introduced by Chandra, Kozen and Stockmeyer in [3]. Alternating Pushdown Automata were further explored in [7], and shown to accept exactly the exponential time languages. As such, they are too strong a model for Conjunctive Grammars. Synchronized Alternating

Pushdown Automata (SAPDA), introduced in [1], are a weakened version of Alternating Pushdown Automata, which, in particular, accept intersections of context-free languages. In [1], SAPDA were proven to be equivalent[1] to CG.

Deterministic context-free languages are a sub-family of context-free languages which can be accepted by a deterministic PDA. In [6], Knuth introduced the notion of $LR(k)$ grammars, and proved their equivalence to deterministic PDA. Through this equivalence, he developed a linear time LR parsing algorithm for deterministic context-free languages, which quickly became the basis of modern-day compilation theory. Furthermore, Knuth proved that $LR(0)$ languages (those which can be parsed with no lookahead) are equivalent to deterministic PDA accepting by empty stack.

In [11], Okhotin presented an extension of Tomita's Generalized LR parsing algorithm [14] for CG. The algorithm utilizes non-deterministic LR parsing, and works for all conjunctive grammars in polynomial time. When applied deterministic context-free languages, the run-time is linear.

In this paper we introduce a sub-family of SAPDA, *Deterministic SAPDA* (DSAPDA), and a sub-family of CG, $LR(0)$ *Conjunctive Grammars*. We prove these sub-families are equivalent, analogously to the context-free case. Furthermore, we present a sophisticated and efficient implementation of DSAPDA, which forms the basis of a deterministic linear time parsing algorithm for $LR(0)$ conjunctive languages. This class of languages *properly* contains the context-free $LR(0)$ languages, thus expanding upon previous results.

2 Preliminaries

In this section, we recall the definitions of CG from [9], and SAPDA, from [1].

2.1 Conjunctive Grammars

Definition 1. *A* Conjunctive Grammar *is a tuple $G = (V, \Sigma, P, S)$, where V, Σ are disjoint finite sets of non-terminal and terminal symbols, $S \in V$ is the designated start symbol, and P is a finite set of rules of the form $A \to (\alpha_1 \, \& \cdots \& \, \alpha_n)$ s.t. $A \in V$ and $\alpha_i \in (V \cup \Sigma)^*$, $i = 1, \ldots, n$. If $n = 1$, we just write $A \to \alpha_1$.*

Definition 2. Conjunctive formulas *over $V \cup \Sigma \cup \{(,), \&\}$ are defined as follows.*

- *All symbols of $V \cup \Sigma$, as well as ϵ, are conjunctive formulas.*
- *If $\mathcal{A}$ and $\mathcal{B}$ are formulas, then $\mathcal{AB}$ is a conjunctive formula.*
- *If $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are formulas, then $(\mathcal{A}_1 \, \& \cdots \& \, \mathcal{A}_n)$ is a conjunctive formula. We call each $\mathcal{A}_i$, $i = 1, \ldots, n$, a* conjunct *of $\mathcal{A}$.*[2]

Definition 3. *For a CG G, the relation of* immediate derivability*, $\Rightarrow_G$, on the set of conjunctive formulas is defined as follows.*

[1] We call two models equivalent if they accept/generate the same class of languages.
[2] Note that this definition is different from Okhotin's definition in [9].

1. *$s_1\ A\ s_2 \Rightarrow_G s_1(\alpha_1\ \&\ \cdots\ \&\ \alpha_n)s_2$, for $A \rightarrow (\alpha_1\ \&\ \cdots\ \&\ \alpha_n) \in P$, and*
2. *$s_1\ (w\&\ \cdots\ \&w)\ s_2 \Rightarrow_G s_1\ w\ s_2$, for $w \in \Sigma^*$,*

where $s_1, s_2 \in (V \cup \Sigma \cup \{(,),\&\})^*$. We refer to 1 and 2 as production *and* contraction *rules, respectively. As usual, $\Rightarrow_G^*$ is the reflexive and transitive closure of $\Rightarrow_G$, and the language of G is $L(G) = \{w \in \Sigma^* \mid S \Rightarrow_G^* w\}$.*

Example 1. ([9, Example 1]) The following CG G generates the non-context-free *multiple agreement* language $\{a^n b^n c^n | n \geq 0\}$. $G = (V, \Sigma, P, S)$, where $V = \{S, A, B, C, D\}$, $\Sigma = \{a, b, c\}$, and P consists of the following rules:

$$S \rightarrow (A\ \&\ C)\ ;\ A \rightarrow aA \mid B\ ;\ C \rightarrow Cc \mid D\ ;\ B \rightarrow bBc \mid \epsilon\ ;\ D \rightarrow aDb \mid \epsilon$$

$L(A) = \{a^m b^n c^n | m, n \geq 0\}$, and $L(C) = \{a^m b^m c^n | m, n \geq 0\}$. Therefore, $L(G) = L(A) \cap L(C) = \{a^n b^n c^n | n \geq 0\}$. For example, abc can be derived by

$$\begin{aligned} S &\Rightarrow (A\ \&\ C) \Rightarrow (aA\ \&\ C) \Rightarrow (aB\ \&\ C) \Rightarrow (abBc\ \&\ C) \Rightarrow (abc\ \&\ C) \\ &\Rightarrow (abc\ \&\ Cc) \Rightarrow (abc\ \&\ Dc) \Rightarrow (abc\ \&\ aDbc) \Rightarrow (abc\ \&\ abc) \Rightarrow abc\ . \end{aligned}$$

2.2 Synchronized Alternating Pushdown Automata

Introduced in [1], SAPDA extend standard PDA by adding the power of intersection. In an SAPDA, transitions are made to a conjunction of states. The model is non-deterministic, i.e., several conjunctions may be possible from a given configuration. If all conjunctions are of one state, the automaton is a standard PDA.

The stack memory of an SAPDA is a tree. Each leaf has a processing head which reads the input and writes to its branch independently. When a conjunctive transition is applied, the stack branch splits into multiple branches, one for each conjunct. When sibling branches empty, they must empty synchronously, i.e. after reading the same portion of the input, and in the same state, after which the computation continues from the parent branch.

Definition 4. *A* synchronized alternating pushdown automaton *is a tuple $A = (Q, \Sigma, \Gamma, \delta, q_0, \bot)$, where δ is a function that assigns to each element of $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ a finite subset of*

$$\{(q_1, \alpha_1) \wedge \cdots \wedge (q_k, \alpha_n) \mid n = 1, 2, \ldots,\ q_i \in Q \text{ and } \alpha_i \in \Gamma^*,\ i = 1, \ldots, n\}\ .$$

Everything else is as in the standard PDA model. Namely, Q is a finite set of states, Σ and Γ are the input and the stack alphabets, respectively, $q_0 \in Q$ is the initial state, and $\bot \in \Gamma$ is the initial stack symbol, see, e.g., [5, pp. 107–112].

Definition 5. *Let A be an SAPDA and let $w \in \Sigma^*$.*

- *The computation of A on w begins in state q_0 and with $\bot$ in the stack.*
- *Each step, a transition is applied to one of the non-empty branches.*
- *Sibling branches that are empty, in state q, and with the same remaining input, are collapsed, and the computation continues from the parent branch in state q.*

– *An accepting computation is one where the entire input is read, and the stack is emptied (i.e., all branches are emptied and collapsed).*

$L(A)$ is the language of all $w \in \Sigma^*$ s.t. A has an accepting computation on w.[3]

Example 2. [2, Example 6.2, pp. 64–65] We construct an SAPDA which accepts the language

$$L_{inf} = \{a^{i_1}b\, a^{i_2}b^2 \;\cdots\; a^{i_n}b^n \;\$\; ba^{i_1}\; ba^{i_2} \;\cdots\; ba^{i_n} \;\$ \mid n \geq 1 \text{ and } i_1, \ldots, i_n \geq 1\} \; .$$

For this, we construct two automata, each in charge of a specific aspect of the language. The first, A_1, checks that the series of bs before the first \$ sign starts at 1 and increases by 1 at each step. The second, A_2, checks that the numbers of as before and after the first \$ match up appropriately. We then define an SAPDA A such that A accepts the intersection of the languages of A_1 and A_2. Following, we present the full construction of A_2. For the construction of A_1 and A, see [2, Example 6.2, pp. 64–65]. The SAPDA $A_2 = (Q_2, \Sigma, \Gamma_2, q_0, \bot, \delta_2)$ is defined as follows. $Q_2 = \{q_0, q_0', q_1, q_1', q_2, q_3, q_e\}$, $\Gamma_2 = \{\bot, a, b\}$, and

(1)	$\delta_2(q_0, a, \bot) = (q_1, a\bot) \wedge (q_0', \bot)$	(2)	$\delta_2(q_0', a, \bot) = (q_0', \bot)$
(3)	$\delta_2(q_0', b, \bot) = (q_0, \bot)$	(4)	$\delta_2(q_0, b, \bot) = (q_0, \bot)$
(5)	$\delta_2(q_0, \$, \bot) = (q_e, \bot)$	(6)	$\delta_2(q_1, a, \bot) = (q_1, a\bot)$
(7)	$\delta_2(q_1, a, a) = (q_1, aa)$	(8)	$\delta_2(q_1, b, a) = (q_1', ba)$
(9)	$\delta_2(q_1', b, b) = (q_1', bb)$	(10)	$\delta_2(q_1', a, b) = (q_2, b)$
(11)	$\delta_2(q_2, a, b) = (q_2, b)$	(12)	$\delta_2(q_2, b, b) = (q_2, b)$
(13)	$\delta_2(q_2, \$, b) = (q_3, b)$	(14)	$\delta_2(q_1', \$, b) = (q_3, b)$
(15)	$\delta_2(q_3, b, b) = (q_3, \epsilon)$	(16)	$\delta_2(q_3, a, b) = (q_3, b)$
(17)	$\delta_2(q_3, a, a) = (q_3', \epsilon)$	(18)	$\delta_2(q_3', a, a) = (q_3', \epsilon)$
(19)	$\delta_2(q_3', b, \bot) = (q_e, \bot)$	(20)	$\delta_2(q_3', \$, \bot) = (q_e, \epsilon)$
(21)	$\delta_2(q_e, a, \bot) = \delta_2(q_e, b, \bot) = (q_e, \bot)$	(22)	$\delta_2(q_e, \$, \bot) = (q_e, \epsilon)$

The automaton recursively opens a new branch for every first a in a series a^{i_j} that it sees. These branches subsequently store $a^{i_j}b^j$ in their stacks, and wait for the \$ sign. After the \$ is read, each branch "counts" to the jth series of a's by popping one b for each b encountered in the input. Thus, a^{i_j} will appear at the top of the stack after j b's have been read. If all a^{i_j} series before and after the \$ match, upon reading the final \$ sign, all branches will be able to empty their stacks, and collapse back to the root. See Figure 1 for sample configurations.

3 Deterministic SAPDA Model Definition

We define the notion of a deterministic SAPDA analogously to the classical definition of a deterministic PDA.

Definition 6. *An SAPDA $A = (Q, \Sigma, \Gamma, q_0, \delta, \bot)$ is* deterministic *if*

[3] For a detailed definition of SAPDA, see [2].

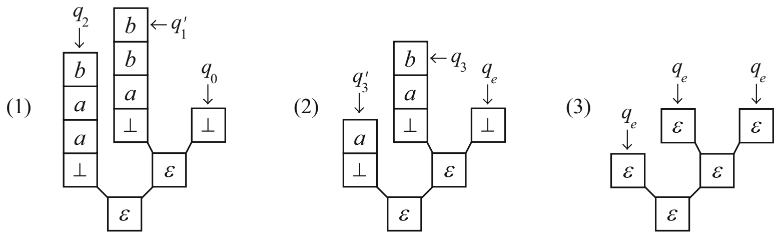


Fig. 1. Configurations of the automaton A_2, (1) after reading $aababb$, (2) after reading $aababb\$ba$, and (3) after reading $aababb\$baaba\$$

- *If* $\delta(q, \sigma, X) \neq \emptyset$ *for some* $\sigma \in \Sigma$*, then* $\delta(q, \epsilon, X) = \emptyset$.
- *For all* $q \in Q, \sigma \in \Sigma \cup \{\epsilon\}, X \in \Gamma, \quad |\delta(q, \sigma, X)| \leq 1$.

By Definition 6, a deterministic SAPDA has at most one computation on any given input word.[4] Note that the automaton A_2 from Example 2 is in fact a deterministic automaton, as is the full automaton for A for L_{inf}, see [2, Example 6.2, pp. 64–65].

3.1 Linear Membership for DSAPDA

We show that the membership problem for DSAPDA is decidable in linear time.

Remark 1. For the purposes of our discussion, we assume the automaton does not have an infinite series of ϵ transitions. As in the classical case, this assumption is not limiting (see [5, Lemma 10.3, p 236]), as ϵ-loops can be detected.

To proceed, we shall need the following notation. Let $A = (Q, \Sigma, \Gamma, q_0, \perp, \delta)$ be a DSAPDA. We denote by N_A the maximal number of branches opened in a single transition, and we denote by M_A the total number of different configurations possible for a branch head, i.e., $M_A = |Q| \times |\Gamma \cup \{\epsilon\}|$.

We consider an implementation model where the computation proceeds in rounds such that in each round, every branch takes one step. Note that this model is equivalent to the one where branches take steps in arbitrary order.

Consider a single stack-branch. As it behaves exactly like a standard deterministic PDA without ϵ-loops, it performs a linear number of steps in the input length. At each step, at most N_A new branches are opened from each existing branch. Therefore, the total number of branches is $O({N_A}^n)$, where n is the number of input letters read. It follows that a DSAPDA can perform an exponential number of steps in the length of the input.

To achieve linear-time membership for DSAPDA languages, we must circumvent the potentially exponential number of stack branches that the automaton can open. To do so, we require the following immediate lemmas.

[4] Up to permutations on the order in which the branches were chosen for transitions.

Lemma 1. *During the computation, at any given time, there are at most M_A different state and stack-symbol configurations among the heads of the stack-branches of the automaton.*

Lemma 2. *If two branches have the same stack-head configuration, then they behave identically on the same input, as long as the stack height does not dip below the initial height of the head.*[5]

By these two lemmas, we do not need the full exponential power of SAPDA to decide membership for DSAPDA. The core concept of the implementation is to execute the minimal number of branches necessary (at most M_A). When a number of branch heads have the same configuration, they are combined to be one head. The computation then continues on the merged branch, as long as its stack is not empty. Once the merged branch empties, the computation continues on the original branches. Thus, at any given computation step, at most M_A branch heads are necessary, and we achieve linear time. Note that this implementation yields a DAG structure rather than a tree, see Figure 2.

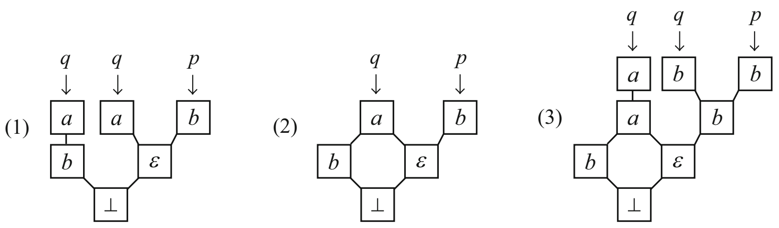


Fig. 2. Example of a stack structure in its initial state, after merging the two leftmost heads, and after applying a transition to the resulting structure

Theorem 1. *The membership problem for DSAPDA is decidable in linear time.*

The correctness of Theorem 1 stems from the observation that during each iteration, at most a constant number of stack symbols are added to the structure, and therefore it is always linearly bounded. For a full proof, see [2, Section 6.2, pp. 67–71].

Remark 2. There is a one-to-one correlation between the full configuration of a DSAPDA and its above compact representation, see [2, Remark 6.8, p. 71].

4 $LR(0)$ Conjunctive Grammars

In this section, we extend the classical notion of an $LR(0)$ grammar (see [5, pp. 248–252]) to CG. We begin with some preliminary definitions.

Definition 7. *Let $G = (V, T, P, S)$ be a CG. The* trace grammar *of G is a* context-free *grammar $G_T = (V, T, P_T, S)$ where P_T is defined as follows:*

[5] This lemma stems from the fact that the automaton is *deterministic.*

1. *$X \to \alpha \in P_T$ for all $X \to \alpha \in P$.*
2. *$X \to \alpha_i \in P_T$ for all $X \to (\alpha_i \& \cdots \& \alpha_n) \in P$ and $i = 1, \dots, n$.*

Rules of type 2 are called projections *of the original conjunctive rules they were obtained from. Applications of these rules are referred to as* conjunct selections*.*

Definition 8. *Let G be a conjunctive grammar, and let G_T be its trace grammar. A derivation in G_T is called a* trace *derivation of G. One-step trace derivations are denoted by $\Rightarrow_T$, and their closure by $\Rightarrow_T^*$.*

Definition 9. *Let G be a conjunctive grammar, and let G_T be its trace grammar. Let $\alpha \Rightarrow^* \mathcal{B}$ be a derivation in G, and let $\alpha \Rightarrow_T^* \beta$ be a trace derivation in G_T. We say that the trace derivation is a* projection *of the full derivation if it follows one of the possible conjunctive paths of the derivation, i.e., the conjunct selections in the trace match the conjunctive rules applied along the path of the full derivation, and the "regular" rules applied in the trace match the non-conjunctive rules applied in the path. If two trace derivations are both projections of the same full derivation, we say that they are* sibling *traces.*[6]

Consider the derivation of the word abc in the CG from Example [1]. The traces $S \Rightarrow A \Rightarrow aA \Rightarrow aB \Rightarrow abBc \Rightarrow abc$ and $S \Rightarrow C \Rightarrow Cc \Rightarrow Dc \Rightarrow aDbc \Rightarrow abc$ are both projections of this derivation, and therefore siblings.

Definition 10. *A derivation $X \Rightarrow^* \mathcal{A}$ is a rightmost(leftmost) derivation if all its projections are rightmost(leftmost) in the classical sense. One-step rightmost(leftmost) derivations are denoted by$\Rightarrow_R$($\Rightarrow_L$), and their closure by$\Rightarrow_R^*$($\Rightarrow_L^*$).*

To facilitate in the construction of a parser, we define an augmented form for CG, which "marks" conjunctive rules using specified new variable symbols. The augmentation process is linear in the number of conjunctive rules in the grammar, and can be implemented as a simple pre-processing step in the construction of a parser.

Definition 11. *Given a conjunctive grammar $G = (V, \Sigma, P, S)$, we define an* augmented *grammar G' by adding the following variables and rules.*

– *We add a new start symbol S', and add a rule $S' \to S$.*
– *Let n be the maximal number of conjuncts in a rule of P. Let m be the number of conjunctive rules in P. For every rule $X \to (\alpha_1 \& \cdots \& \alpha_k) \in P$ that is the j-th conjunctive rule in P, we do the following.*
  - *Add new variables $S_1^j, \dots, S_n^j$ and S^j,*
  - *replace the rule with $X \to S^j$, and add the rule $S^j \to (S_1^j \& \cdots \& S_n^j)$, and the rules $S_i^j \to \alpha_i$ for $i = 1, \dots, k$ and $S_i^j \to \alpha_k$ for $i = k+1, \dots, n$.*

Clearly, for any conjunctive grammar G and its augmentation G', $L(G') = L(G)$. Henceforth, we only consider augmented grammars.

[6] For an inductive definition of projections see [2, Definition 4.3, pp. 32–33].

Following is a very simple example of a conjunctive grammar in augmented form. The example will be used for illustrative purposes, to elucidate the parser construction.

Example 3. The following CG, $G = (V, \Sigma, P, S)$, is an augmented grammar which derives the regular language $\{ab\} \cup \{a^{6n}\$ \mid n \geq 1\}$.

- $V = \{S', S, A, B, S^1, S^1_1, S^1_2\}$, and
- P consists of the following rules:
  $S' \to S$; $S \to ab \mid S^1$; $S^1 \to (S^1_1 \,\&\, S^1_2)$; $S^1_1 \to aaA$; $S^1_2 \to aaaB$; $A \to aaA \mid \$$; $B \to aaaB \mid \$$

We proceed to define the basic building blocks of LR grammars, e.g., items, viable prefixes, and valid prefixes. We define these by applying the classical definitions to the trace grammar.

Definition 12. *Let $G = (V, \Sigma, P, S)$ be a CG. The* set of $LR(0)$ items *of G is the set of classical $LR(0)$ items of the trace grammar G_T,*[7] *i.e.,*

- *$X \to \alpha \cdot \beta$ is an $LR(0)$ item if $X \to \alpha\beta \in P$, and*
- *$S^j \to \cdot\, S^j_i$ and $S^j \to S^j_i\, \cdot$ s.t. $S^j \to (S^j_1 \& \cdots \& S^j_n) \in P$ are $LR(0)$ items.*

Definition 13. *We say that $\gamma \in (V \cup \Sigma)^*$ is a* viable prefix *of G if it is a viable prefix of G_T, i.e., if there is a rightmost trace derivation of G_T of the form $S \Rightarrow^*_{TR} \delta X w \Rightarrow_{TR} \delta\beta w$ s.t. γ is a prefix of $\delta\beta$.*

Definition 14. *We say an item $X \to \alpha \cdot \beta$ is* valid *for a viable prefix γ if there is a trace derivation $S \Rightarrow^*_{TR} \delta X w \Rightarrow_{TR} \delta\alpha\beta w$, $X \to \alpha\beta \in P$, and $\gamma = \delta\alpha$.*

Example 4. Consider the augmented grammar G from Example 3. The derivation $S \Rightarrow S^1 \Rightarrow S^1_1 \Rightarrow aaA \Rightarrow aaaaA$ is a trace derivation of G. Therefore, all prefixes of $aaaaA$ are viable prefixes of G. It follows that the items $A \to a \cdot aA$ and $A \to aa \cdot A$ are valid for the viable prefixes aaa and $aaaa$ respectively.

We proceed to define a DSAPDA that acts as an LR parser for conjunctive languages. To do so, we define a canonical set of item-sets and two functions, *action* and *goto*, which together make up the parsing table for a given grammar. As in the classical case, *goto* recognizes valid items for viable prefixes, and *action* decides which step the automaton takes, based on the set of valid items supplied by *goto*. The main difference from the classical case is that when an item of the form $X \to \cdot\, S^j$ is valid (i.e., a conjunctive rule can be applied), the DSAPDA makes a conjunctive transition and each branch processes one of the conjuncts $S^j_1, \ldots, S^j_n$. This is to avoid conflicts in the parser caused by items from sibling traces.

We begin with *goto*. A *goto* function receives a set of items I and a symbol $X \in V \cup \Sigma \cup \{\epsilon\}$.[8] The function is applied to two types of item sets: *regular* and

[7] Note that the only assumption we make on G is that it is in augmented form.
[8] Note that in the classical *goto* function, $X \in V \cup \Sigma$.

split, see below. When a *goto* function g is applied to regular sets, it behaves exactly as in the classical case, i.e., if I is the set of valid items for some viable prefix γ and $X \neq \epsilon$ then $g(I, X)$ is the set of valid items for viable prefix γX.

When a *goto* function is applied to a split set, it only has ϵ-transitions. Namely, it has n ϵ-transitions to n item-sets, each containing exactly one of the $S^j \to \cdot\ S^j_i$ items, thus separating items from sibling traces. These transitions correlate with the conjunctive transitions in the DSAPDA parser. To accommodate for these "multiple transitions", the *goto* function maps to *sets* of item-sets, as opposed to exactly one item-set, as in the classical case.

Definition 15. *Let I be a set of $LR(0)$ items. We define the* item-closure *of I, denoted $[I]$ as the smallest set of items such that $I \subseteq [I]$, and if $X \to \delta \cdot Y\alpha \in [I]$ and $Y \neq S^j, j = 1, \cdots, m$, then $Y \to \cdot\ \beta \in [I]$ for all $Y \to \beta \in P$.*

This definition is the same as the classical item-closure definition, except that items of the form $X \to \cdot S^j$ are not expanded. This is because their expansions need to be separated, and they are therefore expanded in a separate step.

Definition 16. *A set of items I is* split *if it contains an item of the form $X \to \cdot\ S^j$, yet it does not contain the items $S^j \to \cdot\ S^j_i$. In this case, we also say that S^j* is split *in I. If an item set is not split, it is called* regular*.*

Note that by Definitions 15 and 16, the item-closure of a regular item-set corresponds to the classical item-closure definition.

Definition 17. *A function g is a valid goto function if for each regular item-set I, split item-set J, and symbol $X \in V \cup \Sigma$, the following holds.*

- *$g(I, X) = \{\ [\{Z \to \alpha X \cdot \beta \mid Z \to \alpha \cdot X\beta \in I\}]\ \}$, and $g(I, \epsilon) = \{I\}$.*
- *$g(J, X) = \emptyset$, and $g(J, \epsilon) = \{[J_1], \ldots, [J_n]\}$ where $J_1, \cdots, J_n$ are minimal item-sets such that*
  - *$J \subseteq J_k$ for $k = 1, \ldots, n$, and*
  - *for each j such that S^j is split in J, and for each $i = 1, \ldots, n$, there exists $1 \leq k \leq n$ such that $S^j \to \cdot S^j_i \in J_k$, and for no $i' \neq i$, $S^j \to \cdot S^j_{i'} \in J_k$.*

Note that in transitions from split-sets, exactly one $S^j \to \cdot\ S^j_i$ item from each conjunctive rule in J appears in each resulting item-set. In particular, if J contains only one item of the form $X \to \cdot S^j$, then for $k = 1, \ldots, n$, $J_k = [J \cup\ \{S^j \to \cdot S^j_{i_k}\}\ ]$, for some $1 \leq i_k \leq n$. Furthermore, note that when applied to regular sets, a valid *goto* function behaves exactly like the classical one.

Example 5. Continuing our discussion of the grammar G from Example 3, consider the item-set $I_0 = \{S' \to \cdot S, S \to \cdot ab, S \to \cdot S^1\}$. Note that I_0 is split. Therefore, a valid *goto* function can be defined by $g(I_0, \epsilon) = \{I_1, I_2\}$, where $I_1 = I_0 \cup \{S^1 \to \cdot S^1_1, S^1_1 \to \cdot aaA\}$, and $I_2 = I_0 \cup \{S^1 \to \cdot S^1_2, S^1_2 \to \cdot aaaB\}$. The sets I_1 and I_2 are regular. Therefore, e.g., as in the classical case, $g(I_1, a) = \{S \to a \cdot b, S^1_1 \to a \cdot aA\}$.

Next, we define the set of canonical item-sets of a conjunctive grammar, with respect to a valid *goto* function.

Definition 18. *Let $G = (V, \Sigma, P, S)$ be a conjunctive grammar and g a valid goto function. We define the* canonical collection of item-sets *of G with respect to g to be the smallest set C_g such that*

– *$[\{S' \to \cdot\ S\}] \in C_g$, and*
– *if $I \in C_g$ and $X \in V \cup \Sigma \cup \{\epsilon\}$, then $g(I, X) \subseteq C_g$.*

We denote the item-set $[\{S' \to \cdot\ S\}] \in C_g$ by I_0.

We now proceed to define the notion of an $LR(0)$ grammar.

Definition 19. *We say that an item-set I is* conflict free *if the following holds.*

– *If $X \to \alpha\ \cdot \in I$, then there is no other $Y \to \beta\ \cdot \in I$, $Y \neq X$ or $\alpha \neq \beta$.*
– *If $X \to \alpha\ \cdot \in I$, then there is no item $Y \to \beta \cdot \sigma\gamma \in I$, $\sigma \in \Sigma$.*

The first is a reduce-reduce conflict, and the second is a shift-reduce conflict.

Definition 20. *A conjunctive grammar $G = (V, \Sigma, P, S)$ is $LR(0)$ if there is a valid goto function g for which C_g is conflict free.*

Remark 3. Finding a conflict free grouping is a pre-processing step, and, therefore, does not impact the run-time of the parsing algorithm. Moreover, in Section 5, we will see that for every $LR(0)$ grammar, there exists an equivalent $LR(0)$ grammar, where *any* choice of grouping is guaranteed not to cause conflicts.

Let g be a valid *goto* function for a CG G, and let C_g be the resulting canonical set of item-sets. Together, g and C_g define a finite-state automaton where g is the transition function and C_g is the set of states. The automaton is non-deterministic, as g may have multiple ϵ-transitions from the same state. Let $\hat{g}$ be the standard extension of a non-deterministic transition function to strings and sets of states, see [5, pp. 24–25]. Then, for a set of item-sets $Q \subseteq C_g$ and a string $\gamma \in (V \cup \Sigma)^*$, $\hat{g}(Q, \gamma)$ contains all the item-sets J reachable from some set $I \in Q$ by reading γ.

Figure 3 describes a partial construction of the canonical set of item-sets and a valid *goto* function g for the grammar G from Example 3. Note that the first set, I_0 is split. Therefore, $g(I_0, \epsilon) = \{I_1, I_2\}$ where I_1 and I_2 each contain one of the items $S^1 \to \cdot S_1^1, S \to \cdot S_2^1$. Furthermore, we can see that, e.g.,

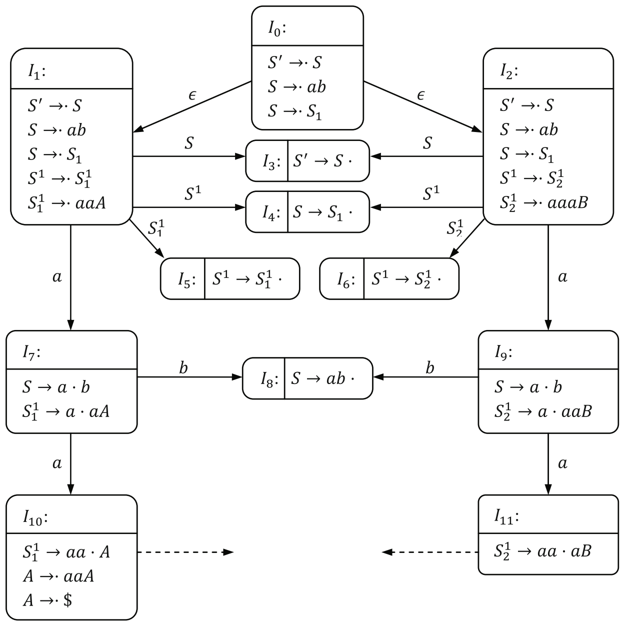
$$\hat{g}(\{I_0\}, a) = \{I_7, I_9\} =$$

$$\{\ \{S \to a \cdot b, S_1^1 \to a \cdot aA\}\ ,\ \{S \to a \cdot b, S_2^1 \to a \cdot aaB\}\ \}\ ,$$

and

$$\hat{g}(\{I_0\}, ab) = I_8 = \{S \to ab\ \cdot\}\ .$$

In the classical construction of an $LR(0)$ item automaton, $goto(I_0, \gamma)$ contains all the valid items for the viable prefix γ. We show that this also holds for a valid *goto* function of a conjunctive grammar. This stems from the fact that for



I_0: $S' \to \cdot S$, $S \to \cdot ab$, $S \to \cdot S_1$
I_1: $S' \to \cdot S$, $S \to \cdot ab$, $S \to \cdot S_1$, $S^1 \to \cdot S_1^1$, $S_1^1 \to \cdot aaA$
I_2: $S' \to \cdot S$, $S \to \cdot ab$, $S \to \cdot S_1$, $S^1 \to \cdot S_2^1$, $S_2^1 \to \cdot aaaB$
I_3: $S' \to S \cdot$
I_4: $S \to S_1 \cdot$
I_5: $S^1 \to S_1^1 \cdot$
I_6: $S^1 \to S_2^1 \cdot$
I_7: $S \to a \cdot b$, $S_1^1 \to a \cdot aA$
I_8: $S \to ab \cdot$
I_9: $S \to a \cdot b$, $S_2^1 \to a \cdot aaB$
I_{10}: $S_1^1 \to aa \cdot A$, $A \to \cdot aaA$, $A \to \cdot \$$
I_{11}: $S_2^1 \to aa \cdot aB$
ϵ
S
S^1
S_1^1
S_2^1
a
b

Fig. 3. Partial construction of the canonical set of item-sets and valid *goto* function

a conjunctive grammar G, the classical $LR(0)$ item-set automaton constructed for its trace, G_T, is the deterministic counterpart of our new item-set automaton construction applied to G. Therefore, the classical item-set automaton's capability to find valid items for viable prefixes is translated to the new construction as well, and we have the following lemma.

Lemma 3. *Let γ be a viable prefix of G and let g be a valid goto function. Then $\bigcup \hat{g}(\{I_0\}, \gamma)$ contains an item $X \to \alpha \cdot \beta$ if and only if $X \to \alpha \cdot \beta$ is valid for γ.*

We can now define a deterministic SAPDA that recognizes the language of an $LR(0)$ grammar G. For each branch, the automaton writes grammar symbols and item-sets from the canonical set of item-sets to its stack, thus keeping track of valid items for the viable prefix it has reduced so far. The transitions of the automaton are determined by the *action* function, and a valid *goto* function g. The initial symbol in the stack is the item-set I_0. The action function receives a current item-set from the top of the stack I, and the next symbol from the input $\sigma \in \Sigma$ (if such a symbol exists), and returns the following:

1. If I contains the item $S' \to S\ \cdot$, the stack is emptied (`accept`).
2. If I is regular and contains an item $X \to \alpha \cdot \sigma\beta$ then σ is shifted onto the stack, and $g(I, \sigma)$ is placed above it (`shift`). Note that $g(I, \sigma)$ returns a single item-set as a non-epsilon transition is applied.

3. If I is regular and contains an item $X \to \alpha\ \cdot$, then the symbols of α and the padding item-sets[9] are removed from the stack, revealing some item set J at the top of the stack. Now, X is written to the stack above J, and then $g(J, X)$ is written on top of that (`reduce`).
4. If I is split, then it is removed from the stack, n new branches are opened, and the n $g(I, \epsilon)$ item sets are put into them, one for each branch (`split`).

We now proceed to show that the automaton does, in fact, accept exactly the language of the grammar. As in the classical case, the automaton attempts to construct a rightmost derivation of the input. To do so, it stores the (tree) prefix of the derivation that it has managed to reduce so far in the stack. At each point, the top symbols of the branches in the stack are the item-sets obtained by applying the *goto* function to this prefix. By Lemma 3, these are exactly the valid items for the prefix, and therefore, they are the candidate derivation rules that can be added next to the rightmost derivation. The automaton continues to shift input symbols onto the stack, until the set of valid items contains an item of the form $X \to \alpha\ \cdot$. This signals that $X \to \alpha$ is the correct choice, and the symbols of α on the stack are replaced with X, thus simulating the reduction. Because the grammar is $LR(0)$, the item-sets are guaranteed to be conflict free, and therefore the automaton is well defined, i.e., it only has one valid transition defined for any given configuration. For a full proof that the rightmost derivation the automaton constructs is correct, see [2, pp. 80–83]. From our automaton construction, we have the following theorem.

Theorem 2. *If a language is generated by an $LR(0)$ conjunctive grammar, then it is accepted by a deterministic SAPDA.*

From Theorems 1 and 2, we obtain the following corollary.

Corollary 1. *Every $LR(0)$ conjunctive language can be parsed in linear time.*

This result extends the context-free $LR(0)$ algorithm, as $LR(0)$ conjunctive languages *properly* contain all finite intersections of classical $LR(0)$ languages. When applied to context-free $LR(0)$ grammars, the parsing algorithm is identical to the classical $LR(0)$ algorithm.

Remark 4. Both classical and conjunctive $LR(0)$ languages are not closed under complement (because the prefix property is not maintained) and under union. However, our linear parser can be modified to work for the boolean closure of conjunctive $LR(0)$ languages as follows. The complement of a language can be determined by running the parser and checking whether the final configuration is accepting or not. As the parser is deterministic, this method will correctly identify the words not in the language. The union of any finite number of languages can be recognized by simply making several parsing runs, one for each language. Thus, we have linear parsing for the boolean closure of $LR(0)$ conjunctive languages, and in particular, the boolean closure of classical $LR(0)$

[9] The padding item-sets are the item-sets pushed to the stack in type 2 transitions.

languages. Furthermore, in Proposition 1 at the end of the following section, we will see that conjunctive $LR(0)$ languages *strictly* contain the boolean closure of context-free $LR(0)$ languages.

5 Constructing an $LR(0)$ Grammar from a DSAPDA

In this section, we address the converse of Theorem 2, i.e., the construction of an $LR(0)$ conjunctive grammar from a deterministic SAPDA.

Theorem 3. *If a language is accepted by a deterministic SAPDA, then it is generated by an* $LR(0)$ *conjunctive grammar.*

Theorem 3 is proved similarly to the classical proof, see e.g., [5, pp. 256–260], by modifying the standard translation of an SAPDA into a CG. The constructed grammar has an important quality whereby for *every* possible valid *goto* function, the canonical set of item-sets constructed is conflict free. For the full construction and proof see [2, Section 6.4, pp. 84–96].

We conclude this section with the following proposition.

Proposition 1. $LR(0)$ *conjunctive languages contain a language that does not belong to the boolean closure of deterministic classical context-free languages.*

The proof of Proposition 1 can be derived directly from the fact that the language L_{inf} from Example 2 is not a finite intersection of context-free languages, see [2, Theorem 6.44, pp. 95–96].

In [11], Okhotin's generalized LR parsing algorithm promises linear-time parsing only for the boolean closure of context-free languages. As such, our result makes a stronger claim regarding the class of linearly-parsable languages.

In [10], Okhotin presents an $LL(k)$ parsing algorithm for conjunctive languages . In the paper, it is stated that it is an open question whether $LL(k)$ conjunctive grammars can generate a language which is not a finite intersection of context-free languages. Therefore, it is an open question whether languages such as L_{inf} can be generated by $LL(k)$ conjunctive grammars. The $LL(k)$ parsing algorithm utilizes a specialized tree-type structure as part of the parsing process. This tree-structure can be viewed as a special case of our SAPDA model, which aligns with the fact that the context-free versions of the $LL(k)$ algorithms is based on Pushdown Automata. Thus, it is reasonable to assume that the parsing algorithm could be modified to work with SAPDA, thus yielding a unified formal approach.

6 Concluding Remarks

We have introduced DSAPDA as a sub-family of SAPDA, and $LR(0)$ CG as a sub-family of CG, and shown that, as in the classical case, acceptance by a DSAPDA is equivalent to generation by an $LR(0)$ CG. This equivalence also forms the basis for a linear time parsing algorithm for an interesting language class comprised of the union closure of conjunctive $LR(0)$ languages.

It would prove interesting to define the notion of $LR(k)$ conjunctive grammars, and specifically $LR(1)$ conjunctive grammars. It would also be interesting to explore uses for DSAPDA based compilers. Two directions seem especially promising. The first is to look for examples where conjunctive grammars give a more succinct representation of a classical $LR(k)$ grammar, therefore leading to more efficient parsing. The second is to find examples of $LR(k)$ conjunctive languages which can be used to describe sophisticated constructs beyond the scope of context free languages. Such examples could prove useful for areas where context free languages have been known to be lacking, such as Natural Language Parsing.

References

1. Aizikowitz, T., Kaminski, M.: Conjunctive grammars and alternating pushdown automata. In: Hodges, W., de Queiroz, R. (eds.) Logic, Language, Information and Computation. LNCS (LNAI), vol. 5110, pp. 30–41. Springer, Heidelberg (2008)
2. Aizikowitz, T.: Synchronized Alternating Pushdown Automata. PhD thesis, Technion – Israel Institute of Technology (2010), `http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi/2010/PHD/PHD-2010-14`
3. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. Journal of the ACM 28(1), 114–133 (1981)
4. Higginbotham, J.: English is not a context-free language. Linguistic Inquiry 15, 119–126 (1984)
5. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (1979)
6. Knuth, D.E.: On the translation of languages from left to right. Information and Control 8, 607–639 (1965)
7. Ladner, R.E., Lipton, R.J., Stockmeyer, L.J.: Alternating pushdown and stack automata. SIAM Journal on Computing 13(1), 135–155 (1984)
8. Langendoen, T.D., Postal, P.M.: English and the class of context-free languages. Computational Linguistics 10(3-4), 177–181 (1984)
9. Okhotin, A.: Conjunctive grammars. Journal of Automata, Languages and Combinatorics 6(4), 519–535 (2001)
10. Okhotin, A.: Top-down parsing of conjunctive languages. Grammars 5(1), 21–40 (2002)
11. Okhotin, A.: LR parsing for conjunctive grammars. Grammars 5(2), 21–40 (2002)
12. Okhotin, A.: A recognition and parsing algorithm for arbitrary conjunctive grammars. Theoretical Computer Science 302, 81–124 (2003)
13. Okhotin, A.: Fast parsing for boolean grammars: A generalization of valiant"s algorithm. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 340–351. Springer, Heidelberg (2010)
14. Tomita, M.: Efficient Parsing for Natural Language: A Fast Algorithm for Practical Systems. Kluwer Academic Publishers, Norwell (1985)

Two-Way Automata versus Logarithmic Space

Christos A. Kapoutsis

LIAFA – Université Paris VII

Abstract. We strengthen previously known connections between the size complexity of two-way finite automata (2FAs) and the space complexity of Turing machines. We prove that

– every s-state 2NFA can be simulated on all poly(s)-long inputs by some poly(s)-state 2DFA if and only if $\mathsf{NL} \subseteq \mathsf{L/poly}$ and
– every s-state 2NFA can be simulated on all $2^{\text{poly}(s)}$-long inputs by some poly(s)-state 2DFA if and only if $\mathsf{NLL} \subseteq \mathsf{LL/polylog}$.

Here, 2DFAs and 2NFAs are the deterministic and nondeterministic 2FAs, NL and $\mathsf{L/poly}$ are the standard space complexity classes, and NLL and $\mathsf{LL/polylog}$ are their counterparts for $O(\log\log n)$ space and poly($\log n$) bits of advice. Our arguments strengthen and extend an old theorem by Berman and Lingas and can be used to obtain variants of the above statements for other modes of computation or other combinations of bounds for the input length, the space usage, and the length of advice.

1 Introduction

The question whether nondeterministic computations can be more powerful than deterministic ones is central in complexity theory. Numerous instantiations have been studied, for a variety of computational models under a variety of resource restrictions. This article investigates the connection between two kinds of such instantiations, those for *Turing machines* (TMs) under *space* restrictions and those for *two-way finite automata* (2FAs) under *size* restrictions.

On the one hand, the question whether nondeterminism makes a difference in space-bounded TMs is among the oldest in complexity theory. Formally, it asks whether there is a bound f with $\mathsf{DSPACE}(f) \not\supseteq \mathsf{NSPACE}(f)$.$^{(1)}$ For $f(n) = n$, this appeared already in [11]. We know that every such bound must be $\Omega(\log\log n)$, since otherwise all languages involved are regular [17,6,1], and that such bounds exist in $\Theta(\log\log n) \cup \Omega(\log n)$ iff $\log\log n$ is already one [13,18]. Since computation in space $\Omega(\log n)$ is more natural than in space $o(\log n)$, research focused on the $\log n$ bound and on the corresponding conjecture that $\mathsf{L} \not\supseteq \mathsf{NL}$. An early observation [2] was that, even if $\mathsf{L} \not\supseteq \mathsf{NL}$, a deterministic TM (DTM) of space $O(\log n)$ may still be able to simulate a nondeterministic one if it is allowed nonuniform behavior. This led to the introduction of the class $\mathsf{L/poly}$ of languages recognizable in space $O(\log n)$ by DTMs accessing poly(n) $\equiv n^{O(1)}$ bits of

⋆ Research funded by a Marie Curie Intra-European Fellowship (PIEF-GA-2009-253368) within the European Union Seventh Framework Programme (FP7/2007-2013).

nonuniform advice [10], and to the stronger conjecture that even $\mathsf{L/poly} \not\supseteq \mathsf{NL}$. Note that, for only $O(\log n)$ advice bits, the resulting conjecture $\mathsf{L/log} \not\supseteq \mathsf{NL}$ is equivalent to $\mathsf{L} \not\supseteq \mathsf{NL}$ [10] and that the classes $\mathsf{DSPACE}(f)/2^{O(f)}$ for varying f have been studied in [8] under the names $\mathsf{NUDSPACE}(f)$.

On the other hand, the question whether nondeterminism makes a difference in size-bounded 2FAs was posed already in [14]. Formally, it asks whether there exist s-state 2NFAs that admit no equivalent 2DFA with poly(s) states. A robust theoretical framework around this question appeared in [12], with the introduction of the complexity classes $\mathsf{2D}$ and $\mathsf{2N}$. The former consists of every family of languages $(L_h)_{h\geq1}$ recognizable by a family of poly(h)-state 2DFAs, while the latter is the corresponding class for 2NFAs. The original question is thus equivalent to whether $\mathsf{2D} \not\supseteq \mathsf{2N}$. We remark that the answer is known to be positive if the 2DFAs must obey certain restrictions [14,15,7,9]; and negative if the 2DFAs are allowed quasi-polynomially many states and the languages are unary [4]. For the general case, the answer has been conjectured to be positive [14,12].

The above two questions are connected via the lengths of the strings necessary to confirm the conjecture $\mathsf{2D} \not\supseteq \mathsf{2N}$. To explain this, let us consider a language family $\mathcal{L} = (L_h)_{h\geq1}$ witnessing the conjecture. Then $\mathcal{L} \in \mathsf{2N}$ but $\mathcal{L} \notin \mathsf{2D}$. The latter means that every family $\mathcal{B} = (B_h)_{h\geq1}$ of poly(h)-state 2DFAs fails to recognize $\mathcal{L}$, in the sense that *at least one* B_h fails to recognize the respective L_h. This is equivalent to saying that every such $\mathcal{B}$ contains *infinitely many* failing B_h (because, if some $\mathcal{B}$ contains only finitely many of them, then replacing those with larger ones that succeed we restore correctness without hurting polynomiality). Of course, a B_h fails to recognize L_h iff it errs on at least one input x_h. Putting everything together, we see that $\mathcal{L} \notin \mathsf{2D}$ iff for every family $\mathcal{B}$ of poly(h)-state 2DFAs there is a family of inputs $\mathcal{X} = (x_h)_{h\geq1}$ such that B_h errs on x_h for infinitely many h. Intuitively, these inputs constitute *hard instances.*

Definition. *A family of* hard instances of $\mathcal{L}$ for $\mathcal{B}$ *is any* $(x_h)_{h\geq1}$ *where there exist infinitely many h such that B_h accepts $x_h \iff x_h \notin L_h$.*

For each $\mathcal{B}$, we can always find hard instances of at most exponential length.

Lemma 0. *Let $\mathcal{L} \in \mathsf{2N}$. If $\mathcal{L} \notin \mathsf{2D}$ then for every family of* poly(h)*-state* 2DFAs *there is a family of* $2^{\mathrm{poly}(h)}$*-long hard instances of $\mathcal{L}$.*

However, in the proof of this lemma,$^{(2)}$ the degree of the polynomial for the 2DFAs lower-bounds the degree of the polynomial exponent for the hard instances. Hence, as the 2DFAs become polynomially larger, the guaranteed hard instances become super-polynomially longer. We thus have no exponential *global* upper bound, for the lengths of hard instances over *all* families of poly(h)-state 2DFAs.

Definition. *A* g-long witness for $\mathsf{2D} \not\supseteq \mathsf{2N}$ *is an* $\mathcal{L} \in \mathsf{2N}$ *such that for every family of* poly(h)*-state* 2DFAs *there is a family of $g(h)$-long hard instances of $\mathcal{L}$.*

So, by Lemma 0, if $\mathsf{2D} \not\supseteq \mathsf{2N}$ then super-exponentially long witnesses (g-long, where $g = \Omega(g')$ for all $g' \in 2^{\mathrm{poly}(h)}$) are guaranteed to exist. But *what about exponentially long witnesses?* Are they guaranteed? Moreover, *what about sub-exponentially long witnesses, e.g., quasi-polynomially or polynomially long ones?*

It has long been known, through a theorem by Berman and Lingas [3], that a polynomially long witness would imply that $\mathsf{L} \not\supseteq \mathsf{NL}$. Here we strengthen that connection: a polynomially long witness would even imply that $\mathsf{L/poly} \not\supseteq \mathsf{NL}$; furthermore, the converse implication is also true.

Theorem 1. $\mathsf{2D} \not\supseteq \mathsf{2N}$ *has* $\mathrm{poly}(h)$*-long witnesses iff* $\mathsf{L/poly} \not\supseteq \mathsf{NL}$.

Moreover, exponentially long witnesses and space $\log\log n$ are similarly linked.

Theorem 2. $\mathsf{2D} \not\supseteq \mathsf{2N}$ *has* $2^{\mathrm{poly}(h)}$*-long witnesses iff* $\mathsf{LL/polylog} \not\supseteq \mathsf{NLL}$.

Here, $\mathsf{LL/polylog} = \mathsf{NUDSPACE}(\log\log n)$ [8] is the class of languages recognizable in space $O(\log\log n)$ by DTMs with $\mathrm{poly}(\log n)$ bits of advice, and NLL is the corresponding class for nondeterministic TMs (NTMs) and no advice.

These two theorems are the most representative ones in a list of variants that can be proved by similar arguments. One group of these variants focus on lengths of other growth rates. E.g., for quasi-polynomial lengths we get:

Theorem 3. *Let* $k \geq 1$ *and* $\varepsilon = 1/k$*. Then* $\mathsf{2D} \not\supseteq \mathsf{2N}$ *has* $2^{O(\log^k h)}$*-long witnesses iff* $\mathsf{DSPACE}(\log^\varepsilon n)/2^{O(\log^\varepsilon n)} \not\supseteq \mathsf{NSPACE}(\log^\varepsilon n)$.

Another group of variants focus on other modes of computation. E.g., consider *alternating* 2FAs and the corresponding complexity class $\mathsf{2A}$. As for $\mathsf{2N}$, we conjecture that $\mathsf{2D} \not\supseteq \mathsf{2A}$ and consider global upper bounds for the lengths of hard instances of an $\mathcal{L} \in \mathsf{2A} \setminus \mathsf{2D}$ over all families of $\mathrm{poly}(h)$-state 2DFAs. We get:

Theorem 4. $\mathsf{2D} \not\supseteq \mathsf{2A}$ *has* $\mathrm{poly}(h)$*-long witnesses iff* $\mathsf{L/poly} \not\supseteq \mathsf{P}$.

(Notice our use of the fact that $\mathsf{AL} = \mathsf{ASPACE}(\log n) = \mathsf{DTIME}\big(\mathrm{poly}(n)\big) = \mathsf{P}$.)

For the most part, the proofs of these theorems elaborate on standard, old ideas [3,12,18]. Perhaps their main value is what they imply for how we approach the two questions being connected. E.g., consider Theorems 1–3 and suppose that indeed $\mathsf{2D} \not\supseteq \mathsf{2N}$. On the one hand, people interested in size-bounded 2FAs can use our theorems to extract evidence about how hard it is for a certain proof strategy towards $\mathsf{2D} \not\supseteq \mathsf{2N}$ to succeed: e.g., a strategy that will eventually deliver a super-exponentially long witness is likely to succeed more easily than one that promises exponentially long witnesses, because the latter implies an additional breakthrough in understanding space-bounded NTMs, while the former does not. (All proof techniques currently available are of the former kind.) On the other hand, people interested in TMs of space $\Omega(\log\log n) \cap O(\log n)$ can find in the $\mathsf{2D}\,\mathsf{v}\,\mathsf{2N}$ question a single unifying setting to work in: separating $\mathsf{2D}$ and $\mathsf{2N}$ with a super-exponentially long witness can be seen as the first step in a gradual approach that sees DTMs and NTMs separate for larger and larger space bounds as improved proof techniques establish shorter and shorter witnesses for $\mathsf{2D} \not\supseteq \mathsf{2N}$.

We conclude this introduction with a different strengthening of the Berman-Lingas Theorem, from [5]: in the case of unary automata $\mathsf{2D} \not\supseteq \mathsf{2N}$ implies $\mathsf{L} \not\supseteq \mathsf{NL}$ *irrespective of the lengths of hard instances.* Our research has been largely motivated by this recent theorem—and our title tries to reflect this.

2 Preparation

For $n \geq 1$, we let $[n] := \{0, \ldots, n-1\}$ and $\lg n := \max(1, \lceil \log_2 n \rceil)$. For S a set, $|S|$ denotes size. If Σ is an alphabet and $x \in \Sigma^*$, then $|x|$ is the length of x; x_i is its ith symbol; x^i is the concatenation of i copies of it; and $\langle x \rangle$ is its *binary encoding*, into $|x|$ blocks of $\lg |\Sigma|$ bits each (under a fixed ordering of Σ); ϵ is the empty string. A (*promise*) *problem* over Σ is a pair $L = (L^+, L^-)$ of disjoint subsets of Σ^*, the *positive* and *negative instances* of L; if $L^+ \cup L^- = \Sigma^*$ then L is a *language*. To *solve* L means to accept all $x \in L^+$ but no $x \in L^-$.

2.1 Machines

We assume familiarity with standard notation for 2FAs and TMs [16]. Our 2FAs consist of a finite *control* and a read-only *input tape*. Our TMs are 2FAs with two extra tapes: a read-only *advice tape* and a read-write *work tape*. Each tape is accessed via a dedicated *two-way head*. A *transducer* is a TM without advice tape but with a write-only *output tape* accessed via a *one-way head*. Details follow.

A (s, σ)-2NFA is any $A = (S, \Sigma, \delta, q_0, F)$ with $|S| = s$, $|\Sigma| = \sigma$, $q_0 \in S$, $F \subseteq S$, and $\delta \subseteq S \times (\Sigma \cup \{\vdash, \dashv\}) \times S \times \{\text{L}, \text{R}\}$, where $\vdash, \dashv \notin \Sigma$ are endmarkers and L, R are directions. An $x \in \Sigma^*$ is presented on the input tape as $\vdash x \dashv$ and is considered accepted if δ allows a computation that starts at q_0 on $\vdash$ and eventually falls off $\dashv$ into a $q \in F$. The *language* of A is the set $L(A) := \{x \in \Sigma^* \mid A \text{ accepts } x\}$. The *binary encoding* of A is the string $\langle A \rangle := \mathtt{0}^s \mathtt{1} \mathtt{0}^\sigma \mathtt{1} uvw$ where u, v, w encode δ, q_0, F with $2s^2(\sigma + 2)$, $\lg s$, and s bits respectively (in the obvious ways, under fixed orderings of S and Σ). Note that $|\langle A \rangle| = O(s^2 \sigma)$.

A (s, σ, γ)-NTM is any $M = (S, \Sigma, \Delta, \Gamma, \delta, q_0, q_f)$ with $|S| = s$, $|\Sigma| = \sigma$, $|\Gamma| = \gamma$, $q_0, q_f \in S$, and $\delta \subseteq S \times (\Sigma \cup \{\vdash, \dashv\}) \times (\Delta \cup \{\vdash, \dashv\}) \times (\Gamma \cup \{\sqcup\}) \times S \times \Gamma \times \{\text{L}, \text{R}\}^3$, where $\sqcup \notin \Gamma$ is the blank symbol. An $x \in \Sigma^*$ and a $y \in \Delta^*$ are presented on the input and advice tapes as $\vdash x \dashv$ and $\vdash y \dashv$ respectively, and are considered accepted if δ allows a computation which starts at q_0, with blank work tape and the input and advice heads on $\vdash$, and eventually falls off $\dashv$ on the input tape into q_f. We say M is in *internal configuration* (q, i, j, w) if its state is q, its advice and work heads are at positions i and j, and its nonblank work tape content is $w \in \Gamma^*$.

For $\mathcal{Y} = (y_m)_{m \geq 0}$ a family of strings over Δ, the *language* of M under $\mathcal{Y}$ is $L(M, \mathcal{Y}) := \{x \in \Sigma^* \mid M \text{ accepts } x \text{ and } y_{|x|}\}$; if $\mathcal{Y}$ is the *empty advice* $(\epsilon)_{m \geq 0}$, we just write $L(M)$. We say $\mathcal{Y}$ is *strong advice* for M if, for all x and $m \geq |x|$, M accepts x and $y_{|x|}$ iff it accepts x and y_m. The *length* of $\mathcal{Y}$ is the function $m \mapsto |y_m|$. A function f is *strong space bound* for M under $\mathcal{Y}$ if, for all x and $m \geq |x|$, all computations on all x and y_m visit at most $f(m)$ work tape cells; if this is guaranteed just for the accepted x and y_m and then only for at least one accepting computation, then f is *weak*. Note that under *strong* advice y_m, M uses space $\leq f(m)$ and is correct on all x with $|x| \leq m$ (and not just $|x| = m$). This deviation from standard definitions is unimportant for advice of length poly(m) or longer (because we can always replace y_m with the concatenation of $y_0, y_1, \ldots, y_m$). For shorter advice, though, it is not clear whether there is a difference. Our theorems need the strong version.

For $\mathcal{G}$ a set of functions, $\mathsf{NSPACE}(f)/\mathcal{G}$ consists of every $L(M, \mathcal{Y})$ where M is a NTM, $\mathcal{Y}$ is *strong* advice for M, with length in $\mathcal{G}$, and f is *strong* space bound for M under $\mathcal{Y}$. If $\mathcal{G} = \{0\}$ then only the empty advice is possible, and we just write $\mathsf{NSPACE}(f)$. Then $\mathsf{NL} := \mathsf{NSPACE}(\log n)$ and $\mathsf{NLL} := \mathsf{NSPACE}(\log\log n)$.

A machine is *deterministic* (2DFA, DTM) if its δ allows at most one computation per input. We define $\mathsf{DSPACE}(f)/\mathcal{G}$, $\mathsf{DSPACE}(f)$, L, LL analogously, and let $\mathsf{L/poly} := \mathsf{DSPACE}(\log n)/\mathrm{poly}(n)$, $\mathsf{LL/polylog} := \mathsf{DSPACE}(\log\log n)/\mathrm{poly}(\log n)$.

A function f is *fully space constructible* if there is a DTM which, on any input x and any advice, halts after visiting exactly $f(|x|)$ work tape cells.

Lemma 1. *Consider a* NTM *M under* strong *advice $\mathcal{Y}$ of length g, obeying a* weak *space bound f. For each length m, there is a $2^{O(f(m)+\lg g(m))}$-state* 2NFA *A_m that agrees with M under $\mathcal{Y}$ on all instances of length at most m.*

If M is deterministic*, then so is A_m. If $\mathcal{Y}$ is* empty*, then A_m also agrees with M under $\mathcal{Y}$ on all negative instances (of any length). If f is* fully space constructible*, then there is a transducer T which, given m (in unary) and the corresponding y_m, computes A_m (in binary) in space $O(f(m) + \lg g(m))$.*

Proof. Let M be a (s, σ, γ)-NTM and $\mathcal{Y} = (y_m)_{m\geq 0}$, g, f as described. Fix m and consider M working on any input (of any length) and on y_m. Let S_m be the set of all internal configurations that can occur with $\leq f(m)$ work tape cells. Then

$$|S_m| \leq s(|y_m|+2) \textstyle\sum_{t=0}^{f(m)} (t+2)\gamma^t = O\big(g(m)f(m)^2\gamma^{f(m)}\big) = 2^{O(f(m)+\lg g(m))} .$$

We let $A_m := (S_m, \Sigma, \delta_m, q_0, F_m)$ where Σ is M's input alphabet, q_0 and F_m are the initial and accepting internal configurations in S_m, and $(c, a, c', d) \in \delta_m$ iff, reading $a \in \Sigma$, M under y_m can change its internal configuration from c to c' moving the input head towards d. Clearly, *if M is deterministic*, then so is A_m.

Now fix any input x (of any length). Let τ_m be the computation tree of M on x and y_m, and τ'_m the computation tree of A_m on x. It should be clear that each branch β in τ_m using $\leq f(m)$ work tape cells is fully simulated in τ'_m by an equally long branch β', which accepts iff β does. In contrast, each β using $> f(m)$ work tape cells is only partially simulated, by a shorter β' which hangs (at a rejecting state). Moreover, the β' of these two cases cover all branches in τ'_m.

Now suppose $|x| = n \leq m$. Let τ_n be the computation tree of M on x and y_n. We know τ_n accepts iff τ_m does (because *$\mathcal{Y}$ is strong*). Thus, if $x \in L(M, \mathcal{Y})$, then τ_n accepts and so does τ_m; hence, some accepting branch β in τ_m uses $\leq f(m)$ work tape cells (because *f is a weak space bound*); hence, its counterpart β' in τ'_m completes the simulation and accepts, too; so, A_m accepts x. Conversely, if $x \notin L(M, \mathcal{Y})$, then τ_n does not accept and neither does τ_m; hence, every branch in τ'_m is nonaccepting (because it either fully simulates a nonaccepting branch of τ_m or hangs after partially simulating one); so, A_m does not accept x.

If $\mathcal{Y}$ is empty, then A_m agrees also on all $x \notin L(M, \mathcal{Y})$ with $n > m$. Because then $y_n = y_m = \epsilon$ implies $\tau_n = \tau_m$, so τ_m does not accept, and neither can τ'_m.

If f is fully space constructible, then $\langle A_m \rangle$ can be computed from 0^m and y_m, as follows. We first mark exactly $f(m)$ work tape cells, by running on 0^m the DTM that fully constructs f. Using these cells and y_m, we then mark another

$\lg s + \lg(|y_m| + 2) + \lg\bigl(f(m) + 2\bigr)$ cells. Now the marked region is as long as the longest description of a $c \in S_m$. This makes it possible to iterate over all $c \in S_m$ or pairs thereof (by lexicographically iterating over all possible strings and discarding non-descriptions). From this point on, computing the bits of $\langle A_m \rangle$ is straightforward. Overall, $O(f(m) + \lg g(m))$ cells will suffice. □

Corollary 1. (a) *For each* NTM M (*under empty advice*) *of weak space* $O(\log n)$ *there is a log-space transducer which, given a length* m (*in unary*), *computes a* 2NFA A_m (*in binary*) *that has* $\text{poly}(m)$ *states and may disagree with* M *only on positive instances longer than* m. *If* M *is deterministic, then so is* A_m.

(b) *For each* NTM M (*under empty advice*) *of weak space* $O(\log\log n)$ *and each length* m *there is a* $\text{poly}(\log m)$*-state* 2NFA A_m *that may disagree with* M *only on positive instances longer than* m. *If* M *is deterministic, then so is* A_m.

2.2 Reductions

Let L_1, L_2 be problems over alphabets Σ_1, Σ_2. A *homomorphic reduction* of L_1 to L_2 [12] is any function $r : \Sigma_1 \cup \{\vdash,\dashv\} \to \Sigma_2^*$ such that $x \in L_1^+ \implies r(\vdash x \dashv) \in L_2^+$ and $x \in L_1^- \implies r(\vdash x \dashv) \in L_2^-$, where $r(\vdash x \dashv) := r(\vdash) r(x_1) \cdots r(x_{|x|}) r(\dashv)$. If there exists such an r, we write $L_1 \leq_\text{h} L_2$. The *expansion* of r is the function $n \mapsto \max\{|r(\vdash x \dashv)| \mid x \in \Sigma_1^* \text{ and } |x| = n\}$. The *ternary encoding* $\langle r \rangle$ of r is the string $\langle r(\vdash)\rangle\#\langle r(a_1)\rangle\#\cdots\#\langle r(a_{\sigma_1})\rangle\#\langle r(\dashv)\rangle$, for $a_1, \dots, a_{\sigma_1}$ a fixed ordering of Σ_1.

Lemma 2 ([12]). *If* $L_1 \leq_\text{h} L_2$ *via a reduction* r *and* L_2 *is solvable by an* s*-state* 2NFA A_2, *then* L_1 *is solvable by a* $2s$*-state* 2NFA A_1. *If* A_2 *is* deterministic, *then so is* A_1. *Moreover, there exists a log-space transducer* T_h *which, given* r (*in ternary*) *and a deterministic* A_2 (*in binary*), *computes* A_1 (*in binary*).

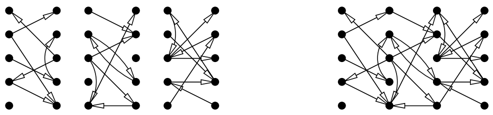
Proof. Let Σ_1 and Σ_2 be the alphabets of L_1 and L_2, and consider r and $A_2 = (Q_2, \Sigma_2, \delta_2, q_2, F_2)$ as in the statement, where $|Q_2| = s$.

On input $x \in \Sigma_1^*$, the new automaton A_1 simulates A_2 on $r(\vdash x \dashv)$ piece-by-piece: on each symbol a on its tape, A_1 does what A_2 would eventually do on the corresponding infix $r(a)$ on its own tape (if a is $\vdash$ or $\dashv$, this corresponding infix is $\vdash r(\vdash)$ or $r(\dashv)\dashv$). This way, each branch β in the computation tree of A_2 on $r(\vdash x \dashv)$ is simulated by a branch in the computation tree of A_1 on x, which is accepting iff β is. Thus, x is accepted iff $r(\vdash x \dashv)$ is. Hence, if $x \in L_1^+$, then $r(\vdash x \dashv) \in L_2^+$ (by the selection of r), hence A_2 accepts, and so does A_1; if $x \in L_1^-$, then $r(\vdash x \dashv) \in L_2^-$ (as before), hence A_2 does not accept, and neither does A_1.

To perform this simulation, A_1 keeps track of the current state of A_2 and the side (L or R) from which A_2 enters the current corresponding infix. Formally, $A_1 := \bigl(Q_2 \times \{\text{L},\text{R}\}, \Sigma_1, \delta_1, (q_2,\text{L}), F_2 \times \{\text{L}\}\bigr)$, where δ_1 is easily derived from the above informal description. E.g., if $a \in \Sigma_1$ then $\bigl((p,\text{L}), a, (q,\text{L}), \text{R}\bigr) \in \delta_1$ iff δ_2 allows on $r(a)$ a computation that starts at p on the leftmost symbol and eventually falls off the rightmost boundary into q (thus entering the following infix from its left side); and $\bigl((p,\text{L}), \dashv, (q,\text{R}), \text{L}\bigr) \in \delta_1$ iff δ_2 allows on $r(\dashv)\dashv$ a computation that starts at p on the leftmost symbol and eventually falls off the leftmost boundary into q (thus entering the preceding infix from its right side).

If A_2 is deterministic, then clearly A_1 is also deterministic. Plus, $\langle A_1\rangle$ can be computed from $\langle r\rangle$ and $\langle A_2\rangle$ in logarithmic space. To see how, let $\sigma_1 := |\Sigma_1|$ and $\sigma_2 := |\Sigma_2|$, and recall that $\langle A_1\rangle = \mathtt{0}^{2s}\mathtt{1}\mathtt{0}^{\sigma_1}\mathtt{1}u_1v_1w_1$ and $\langle A_2\rangle = \mathtt{0}^{s}\mathtt{1}\mathtt{0}^{\sigma_2}\mathtt{1}u_2v_2w_2$, where the u, v, w encode respectively the transition functions, the start states, and the sets of final states. Each part of $\langle A_1\rangle$ can be computed from the corresponding part of $\langle A_2\rangle$ (using no work tape), except for $\mathtt{0}^{\sigma_1}$ and u_1. The former is computed from $\langle r\rangle$, by counting #s. For the latter, we mark $2\lg s+\lg(\sigma_1+2)+3 = O\big(\log(\sigma_1 s)\big)$ work tape cells and use them to iterate over all encodings of tuples $\big((p,d), a, (q,d'), d''\big)$ where $p, q \in [s]$ and $a \in \Sigma_1 \cup \{\vdash,\dashv\}$ and $d, d', d'' \in \{\text{L},\text{R}\}$, outputing 1 bit per tuple. To decide each bit, we locate $\langle r(a)\rangle$ in $\langle r\rangle$ and simulate A_2 on $r(a)$ (or on the appropriate endmarked version, if a is $\vdash$ or $\dashv$) starting at p on side d, until we either exceed $s\cdot|\langle r(a)\rangle|/\lg\sigma_2$ steps or fall off the string; if we fall off side d'' into q and $d' \neq d''$ then we output $\mathtt{1}$, otherwise we output $\mathtt{0}$. This simulation needs only a finite number of pointers into $\langle r\rangle$ and u_2, plus the aforementioned counter. Overall, the space used is logarithmic in $|\langle r\rangle+\langle A_2\rangle|$. □

For $h \geq 1$, the alphabet Γ_h consists of every directed graph with two columns of h nodes each (like the ones on the left, for $h = 5$). Easily, $|\Gamma_h| = 2^{(2h)^2}$. Each



$x \in \Gamma_h^*$ defines a multicolumn graph, derived by identifying adjacent columns (as shown on the right, for the three symbols on the left); if this graph contains a path from its leftmost to its rightmost column, we say x is *live*. We let $\text{TWL}_h := \{x \in \Gamma_h^* \mid x \text{ is live}\}$ (where 'TWL' abbreviates 'TWO-WAY LIVENESS'). Then the family $\text{TWL} := (\text{TWL}_h)_{h\geq 1}$ is 2N-complete [12], because of the following.

Lemma 3 ([12]). *Let A be any s-state 2NFA. Then $L(A) \leq_{\text{h}} \text{TWL}_{2s}$ via a reduction which has expansion $n + 2$ and is log-space constructible from $\langle A\rangle$.*(3)

The *binary encoding* $\langle a\rangle$ of an $a \in \Gamma_h$ is a string of $(2h)^2$ bits that describes a (in the obvious way, under a fixed ordering of the $2h$ nodes). Using these encodings, we can "join" all languages TWL_h into the single binary language

$$\text{TWL JOIN} := \{\mathtt{0}^h\mathtt{1}\langle a_1\rangle\langle a_2\rangle\cdots\langle a_l\rangle\mathtt{1}\mathtt{0}^t \mid h \geq 1 \;\;\&\;\; h \text{ divides } t \;\;\& \\ l \geq 0 \;\;\&\;\; \text{each } a_i \in \Gamma_h \;\;\&\;\; a_1a_2\cdots a_l \text{ is live}\},$$

where the h leading $\mathtt{0}$s determine the interpretation: the leftmost and rightmost $\mathtt{1}$s must be separated by 0 or more $(2h)^2$-long blocks of bits (the "middle bits"), and the multicolumn graph described by these blocks must contain a path from its leftmost to its rightmost column; finally, the number t of trailing $\mathtt{0}$s must be a multiple of h.(4) If this last condition that "h divides t" is replaced by the stronger condition that "every $j \leq h$ divides t" ("Szepietowski-padding" [18]), we get an alternative join, which we call TWL LONG-JOIN.

Let $\textsc{a}_{\textsc{2nfa}} := \{\langle A\rangle\texttt{\#}\langle x\rangle \mid A$ is a 2NFA and A accepts $x\}$ be the *acceptance problem* for 2NFAs, and $\textsc{a}_{\textsc{2dfa}}$ the corresponding problem for 2DFAs. Note that no alphabet is fixed: solving these problems involves checking that the bits after # can be interpreted as an input for the 2FA that is encoded by the bits before #.

Lemma 4. *Let A be an (s,σ)-2NFA. Then $L(A) \leq_{\rm h}$-reduces to each of* $\textsc{a}_{\textsc{2nfa}}$, TWL JOIN, *and* TWL LONG-JOIN *via reductions of expansion $O(s^2\sigma n)$, $O(s^2 n)$, and $2^{O(s)}n$.*(5) *The first two reductions are log-space constructible from $\langle A\rangle$.*

Proof. For $\Sigma = \{a_0, \ldots, a_{\sigma-1}\}$ the alphabet of A, let r_1 be the homomorphism that maps $\vdash$ to $\langle A\rangle\texttt{\#}$, each a_i to the $\lg\sigma$-long binary code of i, and $\dashv$ to ϵ. Then each n-long x is sent to $r_1(\vdash x \dashv) = \langle A\rangle\texttt{\#}\langle x\rangle$, of length $(|\langle A\rangle|{+}1) + n\lg\sigma + 0 = O(s^2\sigma) + n\lg\sigma = O(s^2\sigma n)$, which (clearly) is in $\textsc{a}_{\textsc{2nfa}}$ iff $x \in L(A)$.

For $r : \Sigma \cup \{\vdash,\dashv\} \to \Gamma_{2s}$ the reduction from Lemma 3, let r_2 be the homomorphism that maps $\vdash$ to $\texttt{0}^{2s}\texttt{1}\langle r(\vdash)\rangle$, each a_i to $\langle r(a_i)\rangle$, and $\dashv$ to $\langle r(\dashv)\rangle\texttt{1}\texttt{0}^{2s}$. Then each n-long x is sent to $r_2(\vdash x \dashv) = \texttt{0}^{2s}\texttt{1}\langle r(\vdash)\rangle\langle r(x_1)\rangle \cdots \langle r(x_n)\rangle\langle r(\dashv)\rangle\texttt{1}\texttt{0}^{2s}$, of length $(2s{+}1) + (n{+}2)(4s)^2 + (1{+}2s) = O(s^2 n)$, which is in TWL JOIN iff $r(\vdash)r(x_1)\cdots r(x_n)r(\dashv)$ is live (all other conditions in the definition of TWL JOIN are satisfied), namely iff $r(\vdash x \dashv) \in \textsc{twl}_{2s}$, which is true iff $x \in L(A)$.

Let r_3 be the homomorphism that differs from r_2 only in that it maps $\dashv$ to $\langle r(\dashv)\rangle\texttt{1}\texttt{0}^{\lambda(2s)}$, where $\lambda(2s)$ is the least common multiple of all $j = 1, \ldots, 2s$. Since $\lambda(2s) = 2^{\Theta(s)}$ [18, Lemma, part (b)], now each n-long x is sent to $r_3(\vdash x \dashv)$, of length $(2s{+}1) + (n{+}2)(4s)^2 + (1{+}2^{\Theta(s)}) = 2^{O(s)}n$, which (as before) is in TWL LONG-JOIN iff $r(\vdash)r(x_1)\cdots r(x_n)r(\dashv)$ is live, and thus iff $x \in L(A)$. $\square$

Lemma 5. $\textsc{a}_{\textsc{2nfa}}$ *and* TWL JOIN *are* NL*-complete and* TWL LONG-JOIN $\in$ NLL.

Proof. That $\textsc{a}_{\textsc{2nfa}}$ is NL-complete is well-known; in fact, NL-hardness follows from Corollary 1a and Lemma 4. The NL-hardness of TWL JOIN follows similarly.

To solve TWL JOIN by a NTM in space $O(\log n)$, we work in two stages. First, we deterministically verify the format: we check that there are at least two 1s, count the number h of leading 0s, check that h divides the number of trailing 0s (by scanning and counting modulo h) and that $(2h)^2$ divides the number of "middle bits" (similarly). Then, we nondeterministically verify liveness, by simulating on the "middle bits" the $2h$-state 2NFA solving $\textsc{twl}_h$ [12]. Each stage can be performed in space $O(\log h)$. Since $h \leq n$, TWL JOIN $\in$ NL.

To solve TWL LONG-JOIN we use the same algorithm, but with a preliminary stage [18]. This starts by incrementing a counter from 1 upto the first number, $\tilde{t}$, that does not divide the number t of trailing 0s (divisibility is always tested by scan-and-count, as above). On reaching $\tilde{t}$, we compare it with the number h of leading 0s (scan-and-count, again). If $\tilde{t} \leq h$, we reject; otherwise we continue to the two main stages. Correctness should be clear. The space used in the preliminary stage is $O(\log \tilde{t})$. The two main stages cost $O(\log h)$, as above, which is also $O(\log \tilde{t})$ because we get to them only if $h < \tilde{t}$. Since $\tilde{t} = O(\log t)$ [18, Lemma, part (d)] and $t \leq n$, the total space used is $O(\log\log n)$. $\square$

3 The Berman-Lingas Theorem

The Berman-Lingas Theorem [3, Th. 6] is usually cited as: *if* $\mathsf{L} = \mathsf{NL}$ *then every* s*-state* 2NFA *can be simulated on* $\mathrm{poly}(s)$*-long inputs by some* $\mathrm{poly}(s)$*-state* 2DFA. However, Berman and Lingas actually claim a stronger statement [3, p. 17]:$^{(6)}$

> $\mathsf{L} = \mathsf{NL}$ *iff for each alphabet* Σ *there exists a log-space* DTM T_Σ *which, on input a* 2NFA A *over* Σ (*in binary*) *and a length* m (*in unary*), *outputs a* 2DFA B (*in binary*) *that has* $\mathrm{poly}(sm)$ *states, for* s *the number of states in* A*, and may disagree with* A *only on positive instances longer than* m.

Namely, the promised 2DFA is log-space constructible from the 2NFA and the length bound and, with this into account, the converse is also true. Then, the preceding citation is a corollary for the special case where $m = \mathrm{poly}(s)$.

However, these statements are valid only if Σ is constant. If instead the alphabet grows with s, which is true for some $\mathsf{2N}$-complete problems, then the bound for the states in B might even be exponential in s. To highlight this subtle point, we state and prove the theorem under no assumptions for alphabet size.

Theorem (Berman-Lingas [3]). $\mathsf{L} \supseteq \mathsf{NL}$ *iff there exists a log-space* DTM T *which, on input a* 2NFA A (*in binary*) *and a length* m (*in unary*), *outputs a* 2DFA B (*in binary*) *that has* $\mathrm{poly}(s\sigma m)$ *states, for* s *and* σ *the number of states and symbols in* A*, and may disagree with* A *only on positive instances longer than* m.

Proof. [$\Rightarrow$] Suppose $\mathsf{L} \supseteq \mathsf{NL}$. Then $\mathrm{A}_{\mathrm{2NFA}} \in \mathsf{L}$. Hence, some DTM M solves $\mathrm{A}_{\mathrm{2NFA}}$ under empty advice and in space strongly bounded by $\log n$.

Pick any (s, σ)-2NFA A and length m. By Lemma 4, we know $L(A) \leq_{\mathrm{h}} \mathrm{A}_{\mathrm{2NFA}}$ via a reduction that has expansion $\mu(n) = O(s^2 \sigma n)$ and is constructible from $\langle A \rangle$. Hence (Lemma 2), constructing a 2DFA B for $L(A)$ and lengths $\leq m$ reduces to constructing a 2DFA $\tilde{B}$ for $L(M)$ and lengths $\leq \mu(m)$. This latter construction is indeed possible, by Corollary 1a. We thus employ the series of transductions:

$$\langle A \rangle \texttt{\#0}^m \xrightarrow{T_1} \langle A \rangle \texttt{\#0}^{\mu(m)} \xrightarrow{T_M} \langle A \rangle \texttt{\#} \langle \tilde{B} \rangle \xrightarrow{T_2} \langle r \rangle \texttt{\#} \langle \tilde{B} \rangle \xrightarrow{T_{\mathrm{h}}} \langle B \rangle \,.$$

First, a simple log-space transducer T_1 converts the original input $\langle A \rangle \texttt{\#0}^m$ into $\langle A \rangle \texttt{\#0}^{\mu(m)}$, by doing the algebra. Then the log-space transducer T_M guaranteed by Corollary 1a for M converts $\texttt{0}^{\mu(m)}$ into a 2DFA $\tilde{B}$ that has $\mathrm{poly}\big(\mu(m)\big)$ states and can disagree with M only on positive instances longer than $\mu(m)$. Then the log-space transducer T_2 guaranteed by Lemma 4 for $L(A) \leq_{\mathrm{h}} \mathrm{A}_{\mathrm{2NFA}}$ converts $\langle A \rangle$ into the encoding of the underlying homomorphism r. Finally, the log-space transducer T_{h} from Lemma 2 converts $\langle r \rangle \texttt{\#} \langle \tilde{B} \rangle$ into the desired 2DFA B of $2 \cdot \mathrm{poly}\big(\mu(m)\big) = \mathrm{poly}(s\sigma m)$ states. By the transitivity of log-space transductions, the full algorithm can also be implemented in logarithmic space.

[$\Leftarrow$] Suppose the log-space DTM T of the statement exists. Then $\mathrm{A}_{\mathrm{2NFA}}$ can be reduced to $\mathrm{A}_{\mathrm{2DFA}}$ in logarithmic space, which implies $\mathsf{L} \supseteq \mathsf{NL}$ (since $\mathrm{A}_{\mathrm{2NFA}}$ is NL-hard, $\mathrm{A}_{\mathrm{2DFA}} \in \mathsf{L}$, and L is closed under log-space reductions). The reduction works in two log-space steps. We first run a simple log-space transducer to convert the input $\langle A\rangle\texttt{\#}\langle x\rangle$ into $\langle A\rangle\texttt{\#0}^m\texttt{\#}\langle x\rangle$, where $m := |x| = |\langle x\rangle|/\lg\sigma$, for σ the size of the alphabet of A. We then apply T on $\langle A\rangle\texttt{\#0}^m$ to produce $\langle B\rangle\texttt{\#}\langle x\rangle$, where B a 2DFA that agrees with A on all inputs of length at most m, including x. $\square$

It is now clear that, in the special case where we restrict to inputs of length $m = \text{poly}(s)$, the bound for the states in B is just $\text{poly}(s\sigma)$. Hence, for an A over an alphabet of size $\sigma = 2^{\Omega(s)}$ (e.g., a 2NFA for TWL$_s$), our bound for the states in a 2DFA simulating A on $\text{poly}(s)$-long inputs is just $2^{\text{poly}(s)}$. We stress that this looseness is inherent in the Berman-Lingas argument, and not just in our analysis of it: a larger alphabet for A implies longer encoding $\langle A\rangle$, thus longer inputs to the alleged log-space DTM M for A$_{\text{2NFA}}$, more space available for M on its work tape, more internal configurations, more states in the 2DFA $\tilde{B}$ simulating M, and thus more states in the final 2DFA B simulating $\tilde{B}$.

It is also clear that the above theorem cannot be an equivalence without a constructive relationship between the given 2NFA A and length m and the promised 2DFA B. That is, if in the converse direction we are promised that for each A and m a fitting B simply *exists*, as opposed to it being log-space constructible, then the argument fails. This is an additional obstacle in connecting with the $\mathsf{2D}\,\mathsf{v}\,\mathsf{2N}$ question, which is purely existential: a proof that $\mathsf{2D} \supseteq \mathsf{2N}$ needs no log-space conversion of s-state 2NFAs to equivalent $\text{poly}(s)$-state 2DFAs.

The next theorem removes both of the above dependences, on alphabet size and on log-space constructibility. To remove dependence on alphabet size, we switch to another NL-complete problem; to remove dependence on constructibility, we switch to nonuniform L. The main structure of the argument is similar.

Theorem 1. $\mathsf{L/poly} \supseteq \mathsf{NL}$ *iff for every* 2NFA A *and length* m *there is a* 2DFA B *that has* $\text{poly}(sm)$ *states, for* s *the number of states in* A*, and agrees with* A *on all instances of length at most* m.

Proof. [$\Rightarrow$] Suppose $\mathsf{L/poly} \supseteq \mathsf{NL}$. Then TWL JOIN $\in \mathsf{L/poly}$. Hence, some DTM M solves TWL JOIN under strong advice $\mathcal{Y}$ of length $g(m) = \text{poly}(m)$ in space strongly bounded by $f(m) = \log m$.

Pick any (s,σ)-2NFA A and length m. By Lemma 4, $L(A) \leq_{\text{h}}$ TWL JOIN with expansion $\mu(n) = O(s^2 n)$. Hence (Lemma 2), a $\text{poly}(sm)$-state 2DFA B for $L(A)$ and lengths $\leq m$ exists if there exists a $\text{poly}(sm)$-state 2DFA $\tilde{B}$ for $L(M,\mathcal{Y})$ and lengths $\leq \mu(m)$. Such a $\tilde{B}$ is given by Lemma 1 for M and $\mathcal{Y}$: both $f(\mu(m))$ and $\lg g\big(\mu(m)\big)$ are $O\big(\log(sm)\big)$, and thus $2^{O(f(\mu(m))+\lg g(\mu(m)))} = \text{poly}(sm)$.

[$\Leftarrow$] Suppose that for every s-state 2NFA A and length m there is a $\text{poly}(sm)$-state 2DFA B that agrees with A on all instances of length $\leq m$.

Pick any $L \in \mathsf{NL}$. Let $\varSigma$ be its alphabet and M some NTM that solves it (under empty advice) in space strongly bounded by $\log n$. We know (Corollary 1a) that for each length m there is a $\text{poly}(m)$-state 2NFA A_m over $\varSigma$ that agrees with M on lengths $\leq m$. Applying our assumption to each A_m and m, we find a 2DFA B_m over $\varSigma$ that has $\text{poly}(\text{poly}(m)\cdot m) = \text{poly}(m)$ states and agrees with A_m on lengths $\leq m$, meaning that it decides L correctly on lengths $\leq m$.

Now let $\mathcal{Y} = (y_m)_{m\geq 0} := (\langle B_m\rangle)_{m\geq 0}$ and consider the DTM U over $\varSigma$ which, on input x and advice y, reads y as the encoding of a 2DFA and simulates it on x.

Then $L(U,\mathcal{Y}) = \{x \in \Sigma^* \mid U \text{ accepts } x \text{ and } y_{|x|}\} = \{x \in \Sigma^* \mid B_{|x|} \text{ accepts } x\} = \{x \in \Sigma^* \mid x \in L\} = L$. Note that $\mathcal{Y}$ is *strong* advice: if $|x| \leq m$ then U accepts x and $y_m \Leftrightarrow B_m$ accepts $x \Leftrightarrow x \in L \Leftrightarrow B_{|x|}$ accepts $x \Leftrightarrow U$ accepts x and $y_{|x|}$. Also, $|y_m| = |\langle B_m \rangle| = O(\text{poly}(m)^2 \cdot |\Sigma|) = \text{poly}(m)$ since Σ is constant, and the simulation uses space strongly logarithmic in the number of states in B_m, namely $O\big(\log \text{poly}(m)\big) = O(\log m)$, irrespective of x. Overall, $L \in \mathsf{L/poly}$. $\square$

We remark that the two dependences, on alphabet size and constructibility, do not need to be removed from the Berman-Lingas Theorem simultaneously. Each can also be removed individually, leading to two more variants of the original theorem: one that differs from it only in that the bound for B is poly(sm), and one that differs from Theorem 1 only in that the bound for B is poly$(s\sigma m)$.

Our second extension of the Berman-Lingas Theorem reduces the dependence on the input lengths exponentially, by connecting to the stronger statement that nondeterminism can be nonuniformly removed even on the $\log\log n$ level. The argument is again similar, we just need to switch to the long join of TWL.

Theorem 2. $\mathsf{LL/polylog} \supseteq \mathsf{NLL}$ *iff for every* 2NFA A *and length* m *there is a* 2DFA B *that has* poly$(s \log m)$ *states, for* s *the number of states in* A*, and agrees with* A *on all instances of length at most* m*.*

Proof. [$\Rightarrow$] If $\mathsf{LL/polylog} \supseteq \mathsf{NLL}$ then TWL LONG-JOIN $\in \mathsf{LL/polylog}$, so a DTM M solves TWL LONG-JOIN under strong advice $\mathcal{Y}$ of length $g(m) = \text{poly}(\log m)$ in space strongly bounded by $f(m) = \log\log m$. As above, for any (s,σ)-2NFA A and length m, we know $L(A) \leq_{\mathrm{h}}$ TWL LONG-JOIN via a reduction of expansion $\mu(n) = 2^{O(s)}n$ (Lemma 4). So (Lemma 2), it suffices to find a poly$(s \log m)$-state 2DFA $\tilde{B}$ for $L(M,\mathcal{Y})$ on lengths $\leq \mu(m)$. This is given by Lemma 1, as both $f(\mu(m))$ and $\lg g\big(\mu(m)\big)$ are $O\big(\log(s + \log m)\big)$, and so $2^{O(f(\mu(m)) + \lg g(\mu(m)))} = \text{poly}(s \log m)$.

[$\Leftarrow$] Suppose for each s-state 2NFA A and length m there is a poly$(s \log m)$-state 2DFA B that agrees with A on lengths $\leq m$. As above, we pick any $L \in \mathsf{NLL}$ over some Σ and let M be a NTM solving L in space strongly bounded by $\log\log n$. Corollary 1b guarantees that for each m some poly$(\log m)$-state 2NFA A_m over Σ agrees with M on lengths $\leq m$. Hence (by our assumption), there is also a 2DFA B_m over Σ with poly(poly$(\log m) \cdot \log m) = \text{poly}(\log m)$ states that agrees with A_m on lengths $\leq m$, and thus correctly decides L on these lengths. So, we consider $\mathcal{Y} = (y_m)_{m \geq 0} := (\langle B_m \rangle)_{m \geq 0}$. Under this advice, the same DTM U as above (over the specific Σ) solves L. The advice is *strong* for the same reasons, but its length now is $|y_m| = |\langle B_m \rangle| = O(\text{poly}(\log m)^2 \cdot |\Sigma|) = \text{poly}(\log m)$, causing the space usage to be $O\big(\log \text{poly}(\log m)\big) = O(\log\log m)$. $\square$

A third extension is proved via the variants of TWL JOIN where the padding condition "h divides t" is replaced by "every $j \leq \log^k h$ divides t", for $k \geq 1$.

Theorem 3. $\mathsf{DSPACE}(\log^\varepsilon n)/2^{O(\log^\varepsilon n)} \supseteq \mathsf{NSPACE}(\log^\varepsilon n)$*, where* $\varepsilon = 1/k \leq 1$*, iff for every* 2NFA A *and length* m *there is a* 2DFA B *that has* poly$(s 2^{\log^\varepsilon m})$ *states, for* s *the number of states in* A*, and agrees with* A *on all instances of length at most* m*.*

4 Conclusion

We can now observe that Theorems 1-3 of the previous section are respectively equivalent to Theorems 1-3 of the Introduction. For the pair of Theorems 1 this is shown by the following lemma. Similar lemmas are possible for the other pairs.

Lemma 6. $\mathsf{2D} \not\supseteq \mathsf{2N}$ *has no* $\text{poly}(h)$*-long witnesses iff for every* 2NFA A *and length* m *there is a* 2DFA B *that has* $\text{poly}(sm)$ *states, for* s *the number of states in* A*, and agrees with* A *on all instances of length at most* m*.*

Proof. [$\Rightarrow$] Suppose $\mathsf{2D} \not\supseteq \mathsf{2N}$ has no $\text{poly}(h)$-long witnesses. In particular, TWL is not an h-long witness. Hence, some family $\mathcal{B}' = (B'_h)_{h\geq 1}$ of $\text{poly}(h)$-state 2DFAs admits no h-long hard instances of TWL: i.e., for each family $\mathcal{X} = (x_h)_{h\geq 1}$ of such instances, $\mathcal{B}'$ errs on $\mathcal{X}$ only finitely often. Hence, only finitely many B'_h fail to be correct on all $\leq h$-long instances of TWL_h. Replacing the failing B'_h with larger 2DFAs that do not fail, we get a new family $\mathcal{B} = (B_h)_{h\geq 1}$ of $\text{poly}(h)$-state 2DFAs where B_h decides TWL_h correctly on all $\leq h$-long instances, for all h.

Pick any s-state 2NFA A and length m. Let $\tau := \max(2s, m+2)$. By Lemma 3, we know $L(A) \leq_{\text{h}} \text{TWL}_{2s}$ with expansion $\mu(n) = n+2$. Also, $\text{TWL}_{2s} \leq_{\text{h}} \text{TWL}_\tau$ with expansion $\mu'(n) = n$ (simply map $\vdash, \dashv$ to ϵ, and each $a \in \Gamma_{2s}$ to the $a' \in \Gamma_\tau$ that has only the arrows of a). So, $L(A) \leq_{\text{h}} \text{TWL}_\tau$ with expansion $\mu'(\mu(n))$. So (Lemma 2), a 2DFA B for $L(A)$ on lengths $\leq m$ can be derived from a 2DFA for TWL_τ on lengths $\leq \mu'(\mu(m)) = m+2 \leq \tau$. Such a 2DFA is B_τ, with $\text{poly}(\tau)$ states. Hence, B has $2 \cdot \text{poly}(\tau) \leq 2 \cdot \text{poly}(2s + (m+2)) = \text{poly}(sm)$ states.

[$\Leftarrow$] Suppose for each s-state 2NFA and length m there is a $\text{poly}(sm)$-state 2DFA agreeing on all $\leq m$-long instances. Pick any $\mathcal{L} = (L_h)_{h\geq 1} \in \mathsf{2N}$ and $p \in \text{poly}(h)$. Since $\mathcal{L} \in \mathsf{2N}$, there is a $q \in \text{poly}(h)$ and a family $\mathcal{A} = (A_h)_{h\geq 1}$ of $q(h)$-state 2NFAs solving $\mathcal{L}$. Applying our assumption on each A_h and $p(h)$, we find a 2DFA B_h that agrees with A_h on all $\leq p(h)$-long instances and has $\text{poly}(q(h)p(h)) = \text{poly}(h)$ states. Hence, $(B_h)_{h\geq 1}$ is a family of $\text{poly}(h)$-state 2DFAs that admit no family of $p(h)$-long hard instances of $\mathcal{L}$. ☐

In closing, we suggest some notation that may facilitate discussions like ours. Consider any 2FA family $\mathcal{A} = (A_h)_{h\geq 1}$, any family of promise problems $\mathcal{L} = (L_h)_{h\geq 1}$, and any two function classes $\mathcal{F}, \mathcal{G}$. We say $\mathcal{A}$ is $\mathcal{F}$*-large* if A_h has $\leq f(h)$ states, for some $f \in \mathcal{F}$ and all h; and that $\mathcal{L}$ is $\mathcal{G}$*-long* if $|x| \leq g(h)$, for some $g \in \mathcal{G}$ and all h and $x \in L_h^+ \cup L_h^-$. Then the class $\mathsf{2NSIZE}(\mathcal{F})/\mathcal{G}$ consists of every $\mathcal{G}$-long family of promise problems solvable by some $\mathcal{F}$-large family of 2NFAs —if we just write $\mathsf{2N}/\mathcal{G}$, we mean $\mathcal{F} = \text{poly}(h)$; if we just write $\mathsf{2NSIZE}(\mathcal{F})$, we mean $\mathcal{G}$ contains all functions. We specifically let $\mathsf{2N/poly} := \mathsf{2N}/\text{poly}(h)$ and $\mathsf{2N/exp} := \mathsf{2N}/2^{\text{poly}(h)}$. For 2DFAs, we define the classes $\mathsf{2DSIZE}(\mathcal{F})/\mathcal{G}$, $\mathsf{2D}/\mathcal{G}$, $\mathsf{2DSIZE}(\mathcal{F})$ analogously. Now, Theorems 1, 3, and 2 can be restated as:

$$\begin{aligned}
&\mathsf{2D} \supseteq \mathsf{2N/poly} && \iff && \mathsf{L/poly} \supseteq \mathsf{NL}\\
&\mathsf{2D} \supseteq \mathsf{2N}/2^{O(\log^k n)} && \iff && \mathsf{DSPACE}(\log^\varepsilon n)/2^{O(\log^\varepsilon n)} \supseteq \mathsf{NSPACE}(\log^\varepsilon n)\\
&\mathsf{2D} \supseteq \mathsf{2N/exp} && \iff && \mathsf{LL/polylog} \supseteq \mathsf{NLL}
\end{aligned}$$

where $\varepsilon = 1/k \leq 1$. Note that, next to a TM complexity class, "$/\cdot$" bounds the advice length; next to a 2FA complexity class, it bounds the length of the inputs.

References

1. Alberts, M.: Space complexity of alternating Turing machines. In: Budach, L. (ed.) FCT 1985. LNCS, vol. 199, pp. 1–7. Springer, Heidelberg (1985)
2. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovász, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: Proceedings of FOCS, pp. 218–223 (1979)
3. Berman, P., Lingas, A.: On complexity of regular languages in terms of finite automata. Report 304, Institute of Computer Science, Polish Academy of Sciences, Warsaw (1977)
4. Geffert, V., Mereghetti, C., Pighizzini, G.: Converting two-way nondeterministic unary automata into simpler automata. Theoretical Computer Science 295, 189–203 (2003)
5. Geffert, V., Pighizzini, G.: Two-way unary automata versus logarithmic space. In: Gao, Y., Lu, H., Seki, S., Yu, S. (eds.) DLT 2010. LNCS, vol. 6224, pp. 197–208. Springer, Heidelberg (2010)
6. Hopcroft, J.E., Ullman, J.D.: Some results on tape-bounded Turing machines. Journal of the ACM 16(1), 168–177 (1967)
7. Hromkovič, J., Schnitger, G.: Nondeterminism versus determinism for two-way finite automata: generalizations of Sipser's separation. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 439–451. Springer, Heidelberg (2003)
8. Ibarra, O.H., Ravikumar, B.: Sublogarithmic-space Turing machines, nonuniform space complexity, and closure properties. Mathematical Systems Theory 21, 1–17 (1988)
9. Kapoutsis, C.: Deterministic moles cannot solve liveness. Journal of Automata, Languages and Combinatorics 12(1-2), 215–235 (2007)
10. Karp, R.M., Lipton, R.J.: Some connections between nonuniform and uniform complexity classes. In: Proceedings of STOC, pp. 302–309 (1980)
11. Kuroda, S.: Classes of languages and linear-bounded automata. Information and Control 7, 207–223 (1964)
12. Sakoda, W.J., Sipser, M.: Nondeterminism and the size of two-way finite automata. In: Proceedings of STOC, pp. 275–286 (1978)
13. Savitch, W.J.: Relationships between nondeterministic and deterministic tape complexities. Journal of Computer and System Sciences 4, 177–192 (1970)
14. Seiferas, J.I.: Untitled manuscript, communicated to M. Sipser (October 1973)
15. Sipser, M.: Lower bounds on the size of sweeping automata. Journal of Computer and System Sciences 21(2), 195–202 (1980)
16. Sipser, M.: Introduction to the theory of computation. PWS Publishing Company, Boston, MA (1996)
17. Stearns, R.E., Hartmanis, J., Lewis II, P.M.: Hierarchies of memory limited computations. In: Proceedings of FOCS, pp. 179–190 (1965)
18. Szepietowski, A.: If deterministic and nondeterministic space complexities are equal for $\log \log n$ then they are also equal for $\log n$. In: Monien, B., Cori, R. (eds.) STACS 1989. LNCS, vol. 349, pp. 251–255. Springer, Heidelberg (1989)

Notes

(1)Note the unusual form '$\mathsf{A} \not\supseteq \mathsf{B}$'. Whenever $\mathsf{A} \subseteq \mathsf{B}$, this is of course equivalent to the more familiar '$\mathsf{A} \subsetneq \mathsf{B}$'. Still, we will be studying cases where $\mathsf{A} \subseteq \mathsf{B}$ is false (e.g., $\mathsf{A} = \mathsf{L/poly}$ & $\mathsf{B} = \mathsf{NL}$) and thus '$\mathsf{A} \not\supseteq \mathsf{B}$' is appropriate. We thus stick to it throughout the article, so that every statement is easy to compare with any other. We read and think of '$\mathsf{A} \not\supseteq \mathsf{B}$' as '$\mathsf{A}$ does not cover B' or 'B eludes A'.

$^{(2)}$*Proof* (Lemma 0). The *behavior* of a 2NFA A on a string x is the set of tuples (p, d, q, e) of states p, q and sides $d, e \in \{\text{L,R}\}$ such that A can exhibit a computation that starts at p on the $d$most symbol of x and eventually falls off the $e$most symbol of x into q. Easily, if A has s states, then it can exhibit at most $2^{(2s)^2}$ distinct behaviors. For a 2DFA, this number is $(2s{+}1)^{2s} = 2^{2s \lg(2s+1)}$.

Now suppose $\mathcal{L} \in \mathsf{2N} \setminus \mathsf{2D}$. Pick any 2DFA family $\mathcal{B} = (B_h)_{h \geq 1}$ where B_h has $O(h^b)$ states, for a constant b. Since $\mathcal{L} \notin \mathsf{2D}$, $\mathcal{B}$ admits a family $\mathcal{X} = (x_h)_{h \geq 1}$ of hard instances: B_h errs on x_h for infinitely many h. Since $\mathcal{L} \in \mathsf{2N}$, there is a 2NFA family $\mathcal{A} = (A_h)_{h \geq 1}$ where A_h solves L_h with $O(h^a)$ states, for a constant a.

The number of distinct behaviors that A_h and B_h can exhibit on a particular input are respectively $2^{O(h^{2a})}$ and $2^{O(h^b \log h)}$. Hence, the number g_h of distinct pairs of behaviors of A_h and B_h is $2^{O(h^{2a}+h^b \log h)} = 2^{O(h^c)}$, if $c = \max(2a, b{+}1)$. If x_h is longer than g_h, then it contains two prefixes on which A_h behaves the same and B_h behaves the same, too. Removing the infix of x_h between the right boundaries of these prefixes, we get a shorter input x'_h that neither A_h nor B_h can distinguish from x_h. In particular, B_h errs on x'_h iff it errs on x_h. Repeating this process, we eventually bring the length of x'_h below g_h.

The family of inputs $\mathcal{X}' = (x'_h)_{h \geq 1}$ obtained this way is also a family of hard instances of $\mathcal{L}$ for $\mathcal{B}$, and their lengths are $2^{O(h^c)} = 2^{\text{poly}(h)}$. Notice that, for a family $\mathcal{B}'$ of polynomially larger 2DFAs (i.e., $b' > b$ and $b' \geq 2a$), our argument will return a family of super-polynomially longer hard instances (i.e., $c' > c$ —to be precise, they will be quasi-polynomially longer). □

$^{(3)}$*Proof* (Lemma 3). The reduction maps each $a \in \Sigma \cup \{\vdash, \dashv\}$ to a single symbol $r(a) \in \Gamma_{2s}$ that fully encodes the "behavior of A on a" (as defined in Note 2, see [12] for details). Hence, $|r(\vdash x \dashv)| = |x| + 2$ for all $x \in \Sigma^*$. Constructing $\langle r \rangle$ out of $\langle A \rangle$ is easily done with a finite number of pointers into $\langle A \rangle$. □

$^{(4)}$The suffix $\mathtt{1}\mathtt{0}^t$ and the condition 'h divides t' are redundant. They are included only for symmetry, as the respective suffix and condition are needed in the variants of TWL JOIN (TWL LONG-JOIN and the one described before Theorem 3).

$^{(5)}$More tightly, the first expansion is $O(s^2\sigma) + n \log \sigma$. But the looser $O(s^2\sigma n)$ is simpler and does not harm conclusions, since it is later ([⇒] of B-L Theorem) fed to an unspecified polynomial. Similar looseness is adopted elsewhere, too.

$^{(6)}$The actual statement of [3, Th. 6] is very close to the usual citation given in our text. However, it also includes a pointer to a Remark on p. 17, which explains that the promised 2DFA can be constructed in logarithmic space and that with this observation the theorem becomes an equivalence.

Also note that the second, complete statement in our text uses m as the length bound. The actual statement of [3, Th. 6] uses $s \cdot m$, for s the number of states in the 2NFA. These two statements are equivalent. We have opted for the one that is simpler and facilitates comparison with subsequent statements.

A Polynomial-Time Algorithm for Finding a Minimal Conflicting Set Containing a Given Row[⋆]

Guillaume Blin[1], Romeo Rizzi[2], and Stéphane Vialette[1]

[1] Université Paris-Est, LIGM - UMR CNRS 8049, France
{gblin,vialette}@univ-mlv.fr
[2] DIMI, Università di Udine, Italy
rrizzi@dimi.uniud.it

Abstract. A binary matrix has the *Consecutive Ones Property* (C1P) if there exists a permutation of its columns (i.e. a sequence of column swappings) such that in the resulting matrix the 1s are consecutive in every row. A *Minimal Conflicting Set* (MCS) of rows is a set of rows $\mathcal{R}$ that does not have the C1P, but such that any proper subset of $\mathcal{R}$ has the C1P. In [5], Chauve *et al.* gave a $O(\Delta^2 m^{\max(4,\Delta+1)}(n+m+e))$ time algorithm to decide if a row of a $m \times n$ binary matrix with at most Δ 1s per row belongs to at least one MCS of rows. Answering a question raised in [2], [5] and [25], we present the first polynomial-time algorithm to decide if a row of a $m \times n$ binary matrix belongs to at least one MCS of rows.

1 Introduction

A binary matrix has the *Consecutive Ones Property* (C1P) if its columns can be ordered in such a way that all 1s on each rows are consecutive. Both deciding if a given binary matrix has the C1P and finding the corresponding columns permutation can be done in linear-time [4, 11, 12, 15–17, 19, 22]. A characterization of matrices having the C1P is given in [23]. The C1P of matrices has a long history and it plays an important role in combinatorial optimization, including application fields such as scheduling [1, 13, 14, 28], information retrieval [18], and railway optimization [20, 21, 24] (see [8] for a recent survey).

This paper is devoted to *Minimal Conflicting Sets* (MCS), *i.e.*, minimal sets of rows or columns that prevent the matrix from having the C1P. A *Minimal Conflicting Sets of Rows* (MCSR) (resp. *Minimal Conflicting Sets of Columns* (MCSC)) is a set of rows $\mathcal{R}$ (resp. columns $\mathcal{C}$) of a matrix that does not have the C1P but such that any proper subset of $\mathcal{R}$ (resp. $\mathcal{C}$) has the C1P. Dom [9] has given an algorithm to find a minimum conflicting set in a given matrix. Recent research in comparative genomics has proved MCS to be of particular interest. Indeed, Bergeron *et al.* [2] and Stoye *et al.* [25] have shown how to compute parsimonious evolution scenarios of gene clusters by ranking rows according to

[⋆] Partially founded by ANR Project 2010 JCJC SIMI 2 BIRDS.

their *Conflicting Index* (CI), *i.e.*, the number of MCSR involving a row. In both papers, the problems of efficiently computing the CI of a row and of generating all the MCS of a matrix problems were explicitly raised. Chauve *et al.* [5] gave the first results for those two problems by presenting a $O(\Delta^2 m^{\max(4,\Delta+1)}(n+m+e))$ time algorithm to decide if a row of $m \times n$ binary matrix with at most Δ 1s per row has a positive CI. Note that this algorithm is practical only for small Δ and Chauve *et al.* left as an open problem the question of whether there exists a polynomial-time algorithm to decide if a row has a positive CI. In this paper we give a positive answer to this open problem by combining characterization of matrices having the C1P with graph pruning techniques.

This paper is organized as follows. In Section 2, we recall basic definitions and formally introduce the problem we are interested in. We give in Section 3 a polynomial-time algorithm to decide if a row has a positive CI, and propose in Section 4 some natural extensions. Due to space constraint, most proofs are omitted.

2 Preliminaries

We assume readers have basic knowledge about graph theory [7] and we shall thus use most conventional terms of graph theory without defining them (we only recall basic definitions). Let $G = (V, E)$ be a graph. The *neighborhood* of a vertex $v \in V$ is the set $N(v) = \{u : \{u, v\} \in E\}$. Two distinct vertices $u, v \in V$ are called *twins* if they have the same neighborhood, *i.e.*, $N(u) = N(v)$. For any $V' \subseteq V$, we denote by $G[V']$ the *subgraph of* G *induced by* V' with the additional property that all isolated vertices have been deleted (whereas the latter requirement is non-standard it will prove useful to simplify the presentation). A *path* from vertex u to vertex v is abbreviated to a uv-path. Finally, for any path p in G, we let $V(p) \subseteq V$ stand for the set of all vertices involved in p.

A matrix M is *simple* if it does not contain two identical columns or rows, and *simplifying* a matrix is the (polynomial-time) process of deleting identical rows and columns. In the sequel, we assume any matrix to be simplified. A $(0, 1)$-matrix is a matrix in which each entry is either zero or one. Let M be a $m \times n$ $(0, 1)$-matrix. Its corresponding vertex-colored bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$ is defined as follows: for every row (resp. column) of M there is a black (resp. white) vertex in $\mathcal{R}$ (resp. $\mathcal{C}$), and there is an edge between a black vertex v_i and a white vertex v_j, *i.e.*, an edge between the vertices that correspond to the i^{th} row and the $j^{\text{th}th}$ column of M, if and only if $M[i, j] = 1$. Equivalently, M is the reduced adjacency matrix of $G(M)$. We shall usually write $\mathcal{R} = \{r_i : 1 \leq i \leq m\}$ and $\mathcal{C} = \{c_j : 1 \leq j \leq n\}$. In the sequel, we shall speak indistinctly about binary matrices and their corresponding vertex-colored bipartite graphs. An *asteroidal triple* is an independent set of three vertices such that each pair is joined by a path that avoids the neighborhood of the third. Tucker [27] has proved that if a $(0, 1)$-matrix contains an asteroidal triple then it does not have the C1P. Furthermore, Tucker has given a complete characterization of matrices containing asteroidal triples.

Theorem 1 ([27], Theorem 9). *A* $(0,1)$-*matrix has the C1P if and only if it contains none of the matrices* M_{I_k}, M_{II_k}, M_{III_k} *(*$k \geq 1$*),* M_{IV} *and* M_V *depicted in Figure 1.*

Write $\mathcal{T} = \{M_{I_k}, M_{II_k}, M_{III_k}, M_{IV}, M_V\}$. Let M be a $(0,1)$-matrix. According to Theorem 1, for any MCSR $\mathcal{R}_T$ of M, $G(M)[\mathcal{R}_T \cup \mathcal{C}]$ contains at least one Tucker configuration $T = (\mathcal{R}_T, \mathcal{C}_T, E') \in \mathcal{T}$, and for any $\mathcal{R}'_T \subsetneq \mathcal{R}_T$, $G(M)[\mathcal{R}'_T \cup \mathcal{C}]$ has the C1P, *i.e.*, it does not contain a Tucker configuration. A similar observation can be done for MCSC. For the sake of brevity, any Tucker configuration contained in an MCSR (or MCSC) will be said to be *responsible* for this MCSR (or MCSC).

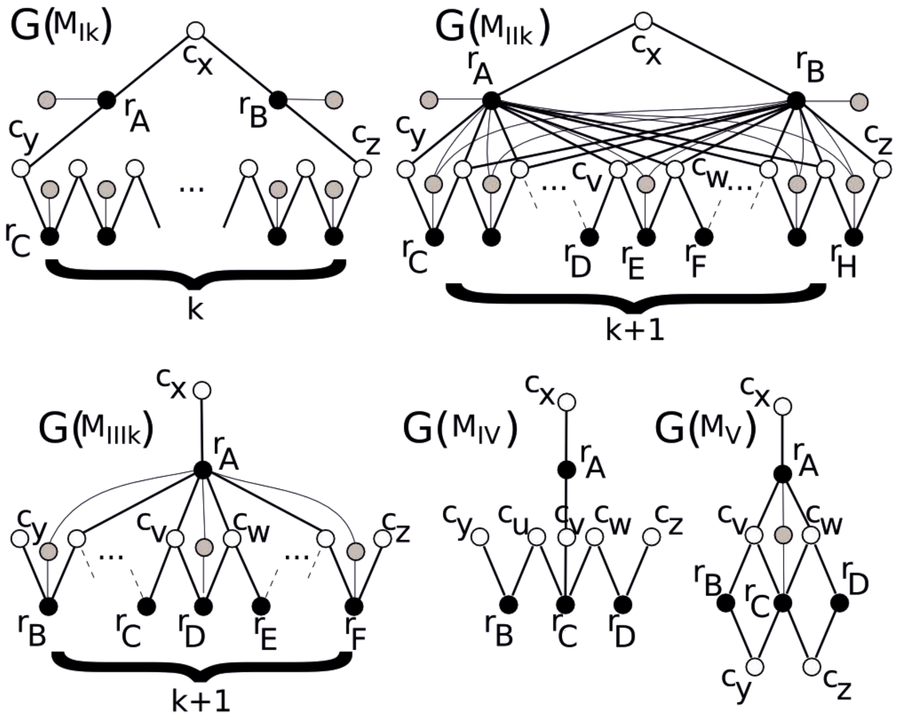


Fig. 1. Forbidden bipartite graphs [27]. Black (resp. white) vertices correspond to rows (resp. columns) of the corresponding matrices. Gray vertices and light edges are not part of the Tucker configurations but represent the extra columns that our algorithm will report. For $G(M_{I_k})$, any triple of white vertices forms an asteroidal triple. For all other forbidden structures, there are exactly one asteroidal triple (c_x, c_y, c_z).

Following our previous work on Tucker forbidden structures [3], our algorithm is based on shortest paths and two graph pruning techniques (graph pruning techniques were introduced by Conforti *et al.* [6]). Let us define the `clean` and `anticlean` pruning operations. Let M be a binary matrix and $G(M) = (\mathcal{R}, \mathcal{C}, E)$ be the corresponding vertex-colored bipartite graph. For any vertex $v \in \mathcal{R}$ (resp. $v \in \mathcal{C}$), $\texttt{clean}(v)$ results in the graph $G(M)[\mathcal{R} \cup (\mathcal{C} \setminus N(v))]$ (resp. $G(M)[(\mathcal{R} \setminus N(v)) \cup \mathcal{C}]$). In other words, $\texttt{clean}(v)$ results in a graph where any neighbor of v has been deleted. For any node $v \in \mathcal{R}$ (resp. $v \in \mathcal{C}$), $\texttt{anticlean}(v)$ results in the graph $G(M)[\mathcal{R} \cup (\mathcal{C} \setminus \{u : u \notin N(v)\})]$ (resp. $G(M)[(\mathcal{R} \setminus \{u : u \notin N(v)\}) \cup \mathcal{C}]$). In other words, $\texttt{anticlean}(v)$ results in a graph where any node that does not belong to the same partition nor the neighborhood of v has been deleted. By abuse of notation, we shall write $\texttt{clean}(u_1, u_2, \ldots, u_k)$ for the sequence $\texttt{clean}(u_1), \texttt{clean}(u_2), \ldots, \texttt{clean}(u_k)$ (a similar abuse will be used for `anticlean`).

Remark 1. It is of particular importance to note that we shall always consider that vertices given as inputs to our algorithms will never be affected (*i.e.*, deleted) by the `clean` and `anticlean` operations.

3 Finding an MCSR Involving a Given Row

We present in this section a polynomial-time algorithm for reporting (if it exists) an MCSR involving a given row. Our main result can be stated as follows.

Proposition 1. *Let M be $m \times n$ $(0,1)$-matrix. For any row r of M, deciding whether there exists an MCSR involving row r is solvable in $O(m^6 n^5 (m+n)^2 \log(m+n))$ time.*

To prove Proposition 1, we provide a sequence of polynomial-time algorithms for finding a minimal Tucker configuration of a given type $T \in \mathcal{T} = \{M_{I_k}, M_{III_k}, M_{II_k}, M_{IV}, M_V\}$ (in this particular order) responsible for an MCSR involving a given row (if it exists). The following easy lemma will prove to be extremely useful in the sequel.

Lemma 1. *Let $T = (\mathcal{R}_T, \mathcal{C}_T, E_T)$ be a Tucker configuration responsible for an MCSR involving a given row r in $G(M) = (\mathcal{R}, \mathcal{C}, E)$. Then $\mathcal{R}_T$ is an MCSR involving r and there is no smaller Tucker configuration – in terms of number of rows (or black nodes) – in $G(M)[\mathcal{R}_T \cup \mathcal{C}]$.*

3.1 M_{I_k} Tucker Configurations

Proposition 2. *Let M be a $(0,1)$-matrix with corresponding vertex-colored bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, and r be any row of M. Finding (if it exists) a minimum cardinality $\mathcal{R}_T \subseteq \mathcal{R}$ responsible for an MCSR involving row r such that $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{I_k})$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ and some $k \geq 1$ is a $O(m^4 n^4)$ time procedure.*

Observe that M_{I_k} is a hole (a chordless cycle of length at least 6), and hence without loss of generality we associate r to r_A in $G(M_{I_k})$ (see Figure 1). We need to consider three cases: $k = 1$, $k = 2$ and $k > 2$. We first try to find

Algorithm 1. Check-$M_{I_k}(c_x, c_y, r_A, r_B, r_C)$

```
Require: A bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, three black vertices $r_A, r_B, r_C \in \mathcal{R}$ ($r$
    identified to $r_A$) and two white vertices $c_x, c_y \in \mathcal{C}$ such that $(r_C, c_y, r_A, c_x, r_B)$ is a
    path in $G(M)$. It is assumed that $G(M)$ does not contain any $G(M_{I_1})$ or $G(M_{I_2})$
    involving row $r$.
Ensure: Return $\mathcal{R}_T \subseteq \mathcal{R}$ such that $G(M_{I_k}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ is an
    MCSR involving row $r$, or return the failure message “NO” if such a configuration
    does not exist.
 1: if $N(r_A) \cap N(r_B) \cap N(r_C) \neq \emptyset$ then
 2:     return “NO”
 3: end if
 4: clean($c$) for all $c \in N(r_A) \setminus N(r_B)$
 5: clean($c$) for all $c \in N(r_A) \setminus N(r_C)$
 6: clean($r_A, c_x, c_y$)
 7: delete vertex $r_A$
 8: if there exists a $r_B r_C$-path in the pruned graph then
 9:     let $P$ be a shortest $r_B r_C$-path in the pruned graph
10:     return return $\{r_A\} \cup \{r_i : r_i \in V(P) \cap \mathcal{R}\}$
11: else
12:     return “NO”
13: end if
```

some $G(M_{I_1}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ involving row r using any brute-force algorithm. If we succeed, we are done since any proper subset of $\mathcal{R}_T$ – of size at most 2 – cannot contain any other Tucker configuration. Otherwise, using any brute-force algorithm, we try to find some $G(M_{I_2}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ involving row r with the additional property that there do not exist $\mathcal{R}'_T \subsetneq \mathcal{R}_T$ and $\mathcal{C}'_T \subseteq \mathcal{C}$ such that $G(M_{III_1}) = G(M)[\mathcal{R}'_T \cup \mathcal{C}'_T]$. This latter additional constraint is necessary and sufficient since $G(M_{III_1})$ is the only smaller Tucker configuration involving row r that could occur in $G(M)$. If both tries failed, we turn to $k > 2$ and apply Algorithm 1 for every tuple of parameters (c_x, c_y, r, r_B, r_C), where $c_x, c_y \in \mathcal{C}$, $r_B, r_C \in \mathcal{R}$, and (r_C, c_y, r, c_x, r_B) is a path in $G(M)$. Among the non-failure answers (if any), we return the smallest one.

Lemma 2. *If there exist an MCSR $\mathcal{R}_T \subseteq \mathcal{R}$ with $\{r_A = r, r_B, r_C\} \subseteq \mathcal{R}_T$ such that $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{I_k})$ for some $k > 2$ and some $\mathcal{C}_T \subseteq \mathcal{C}$ with $\{c_x, c_y\} \subseteq \mathcal{C}_T$, then Algorithm 1 for parameters $(c_x, c_y, r_A = r, r_B, r_C)$ finds it.*

We now turn to evaluating the time complexity of one call to Algorithm 1. Checking that $N(r_i) \cap N(r_B) \cap N(r_C)$ is empty is a $O(n)$ time procedure. Cleaning any white vertex can be done in $O(m)$ time and cleaning r_A can be done in $O(n)$ time. Using a BFS search, finding a shortest $r_B r_C$-path is $O(n + m + mn)$ time. Summing up, the total time complexity of Algorithm 1 is $O(mn)$.

Correctness of Proposition 2 follows from Lemma 2. What is left is to prove the total time complexity. According to Lemma 2, for any row r, we can call Algorithm 1 for parameters $(c_x, c_y, r_A = r, r_B, r_C)$ to find an MCSR (if it exists) involving row r. There are $O(m^2 n^2)$ such tuples, and hence we have a $O(m^3 n^3)$

time procedure for $k > 2$. As for $k = 1$ and $k = 2$, a brute-force algorithm yields a $O(m^4n^4)$ time procedure, the dominant term in our approach for $G(M_{I_k})$ Tucker configurations.

3.2 M_{III_k} Tucker Configurations

We assume in this subsection that there does not exist a $G(M_{I_k})$ Tucker configuration in $G(M)$ responsible for an MCSR involving a given row r.

Proposition 3. *Let M be a $(0, 1)$-matrix with corresponding vertex-colored bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, and r be any row of M. Assuming that there does not exist a $G(M_{I_k})$ in $G(M)$ responsible for an MCSR involving row r, finding (if it exists) a minimum cardinality $\mathcal{R}_T \subseteq \mathcal{R}$ responsible for an MCSR involving row r such that $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{III_k})$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ and some $k \geq 1$ is a $O(m^5n^5(m + n)^2 \log(m + n))$ time procedure.*

If such a $G(M_{III_k})$ Tucker configuration exists and is responsible for an MCSR involving row r, then r can be any of the black vertices of $G(M_{III_k})$. However, thanks to symmetries, it is enough to suppose that row r is identified to r_A, r_D or r_F in $G(M_{III_k})$.

Our algorithm is as follows. If we don't succeed in finding some $G(M_{I_k})$ Tucker configuration responsible for an MCSR involving row r (see Subsection 3.1), we look for some $T = G(M_{III_1}) = (\mathcal{R}_T, \mathcal{C}_T, E_T)$ Tucker configuration involving row r (brute-force algorithm). If such a $G(M_{III_1})$ Tucker configuration exists, $\mathcal{R}_T$ is certainly an MCSR (involving row r). If we fail, we call Algorithm 2 for every tuple of arguments $(c_x, c_y, c_z, r, r_B, r_F)$ with $r_B, r_F \in \mathcal{R}$ and $c_x, c_y, c_z \in \mathcal{C}$, and next call Algorithm 3 for every tuple of arguments $(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r, r_E, r_F)$ with $r_A, r_B, r_C, r_E, r_F \in \mathcal{R}$ and $c_v, c_w, c_x, c_y, c_z \in \mathcal{C}$. Among the non-failure solutions, we return the smallest one.

Lemma 3. *If there exists an MCSR $\mathcal{R}_T \subseteq \mathcal{R}$ involving row r (identified to r_A or r_B) such that $\{r_A, r_B, r_F\} \subseteq \mathcal{R}_T$ and $\{c_x, c_y, c_z\} \in \mathcal{C}_T$, and $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{III_k})$ for some $k > 1$ and some $\mathcal{C}_T \subseteq \mathcal{C}$, then Algorithm 2 for arguments $(c_x, c_y, c_z, r_A, r_B, r_F)$ finds it.*

Lemma 4. *If there exists an MCSR $\mathcal{R}_T \subseteq \mathcal{R}$ involving row r (identified to r_D) such that $\{r_A, r_B, r_C, r_D, r_E, r_F\} \subseteq \mathcal{R}_T$ and $\{c_v, c_w, c_x, c_y, c_z\} \in \mathcal{C}_T$, and $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{III_k})$ for some $k > 1$ and some $\mathcal{C}_T \subseteq \mathcal{C}$, then Algorithm 3 for arguments $(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r, r_E, r_F)$ finds it.*

We now turn to evaluating the time complexity of Algorithm 3 (the time complexity of Algorithm 2 is clearly negligible with that of Algorithm 3). There are $O(m^5n^5)$ calls to Algorithm 3, and hence the whole procedure (summing up all calls to Algorithm 3) is $O(m^5n^5(m+n)^2 \log(m+n))$ time. As for the exhaustive search for $G(M_{III_1})$ Tucker configurations, it is $O(m^3n^4)$ time. Therefore, the algorithm, as a whole, is $O(m^5n^5(m + n)^2 \log(m + n))$ time. Proposition 3 is proved.

Algorithm 2. Check-$M_{III_k}(c_x, c_y, c_z, r_A, r_B, r_F)$

Require: A bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, three black vertices $r_A, r_B, r_F \in \mathcal{R}$ and three white vertices $c_x, c_y, c_z \in \mathcal{C}$ such that $r_A \subseteq N(c_x)$, $r_B \subseteq N(c_y)$, and $r_F \subseteq N(c_z)$. Row r is identified to r_A or r_B. It is assumed that $G(M)$ does not contain any $G(M_{I_k})$ or $G(M_{III_1})$ Tucker configuration involving row r.

Ensure: Return $\mathcal{R}_T \subseteq \mathcal{R}$ such that where $G(M_{III_k}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ is a (row-) minimal MCSR involving row r if it exists, or the failure message "NO" if such a configuration does not exist.

```
1: clean($c_x, c_y, c_z$)
2: clean($c$) for all $c \notin N(r_A)$
3: anticlean($r_A$)
4: remove vertex $r_A$
5: **if** there exists a $r_B r_F$-path in the pruned graph **then**
6:    let $P$ be a shortest $r_B r_F$-path in the pruned graph
7:    **return** return $\{r_A\} \cup \{r : r \in V(P) \cap \mathcal{R}\}$
8: **else**
9:    **return** "NO"
10: **end if**
```

Algorithm 3. Check-$M_{III_k}(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r_D, r_E, r_F)$

Require: A bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, six black vertices $r_A, r_B, r_C, r_D, r_E, r_F \in \mathcal{R}$ and five white vertices $c_v, c_w, c_x, c_y, c_z \in \mathcal{C}$ such that $r_A \subseteq N(c_x) \cap N(c_v) \cap N(c_w)$, $r_B \subseteq N(c_y)$, $r_F \subseteq N(c_z)$, $r_C \subseteq N(c_v)$, $r_D \subseteq N(c_v) \cap N(c_w)$, and $r_E \subseteq N(c_w)$. Row r is identified to r_D. It is assumed that $G(M)$ does not contain any $G(M_{I_k})$ or $G(M_{III_1})$ Tucker configuration involving row r.

Ensure: Return $\mathcal{R}_T \subseteq \mathcal{R}$ such that $G(M_{III_k}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ is a (row-) minimal MCSR involving row r, or the failure message "NO" if such a configuration does not exist.

```
1: **if** $N(r_C) \cap N(r_D) \cap N(r_E) \neq \emptyset$ **or** $(N(r_C) \cup N(r_D) \cup N(r_E)) \setminus N(r_A) \neq \emptyset$ **then**
2:    **return** "NO"
3: **end if**
4: clean($c$) for all $c \in N(r_D)$
5: clean($c_v, c_w, c_x, c_y, c_z$)
6: clean($c$) for all $c \notin N(r_A)$
7: anticlean($r_A$)
8: remove the node $r_A$
9: **if** there exists a $r_B r_F$-path using $r_D$ in the pruned graph **then**
10:    let $P$ be a shortest such $r_B r_F$-path in the pruned graph
11:    **return** return $\{r_A\} \cup \{r | r \in V(P) \cap \mathcal{R}\}$
12: **else**
13:    **return** "NO"
14: **end if**
```

3.3 M_{II_k} Tucker Configurations

We assume in this subsection that there does not exist a $G(M_{I_k})$ nor a $G(M_{III_k})$ Tucker configuration in $G(M)$ responsible for an MCSR involving a given row r.

Proposition 4. *Let M be a $(0,1)$-matrix with corresponding vertex-colored bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, and r be any row of M. Assuming that there does not exist a $G(M_{I_k})$ in $G(M)$ responsible for an MCSR involving row r, finding (if it exists) a minimum cardinality $\mathcal{R}_T \subseteq \mathcal{R}$ responsible for an MCSR involving row r such that $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{II_k})$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ and some $k \geq 1$ is a $O(m^6 n^5 (m+n)^2 \log(m+n))$ time procedure.*

Notice that if such a $G(M_{II_k})$ Tucker configuration does exist and is responsible for an MCSR involving row r then r can be any of the black vertices of $G(M_{II_k})$ (see Figure 1). However, thanks to symmetries, it is enough to suppose that row r is identified to r_A, r_C or r_E in $G(M_{II_k})$ (all other possibilities are equivalent up to a straightforward renaming). Although at first odd, it is also crucial for correctness to assume that no $G(M_{I_k})$ is responsible in $G(M)$ for an MCSR involving row r.

Our algorithm is as follows. If we don't succeed in finding some $G(M_{I_k})$ Tucker configuration responsible for an MCSR involving row r (see Subsection 3.1), we next look for some $G(M_{II_1}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ Tucker configuration involving row r using any brute-force algorithm. If we succeed, we are done since any proper subset of $\mathcal{R}_T$ – of size at most 3 – cannot contain any other Tucker configuration. Otherwise, we use a three-step procedure. We first call Algorithm 4 for every tuple $(c_x, c_y, c_z, r_A, r_B, r_C, r_H)$ with $r_A = r, r_B, r_C, r_H \in \mathcal{R}$ and $c_x, c_y, c_z \in \mathcal{C}$, and next for every tuple $(c_x, c_y, c_z, r_A, r_B, r_C, r_H)$ with $r_A, r_B, r_C = r, r_H \in \mathcal{R}$ and $c_x, c_y, c_z \in \mathcal{C}$. Finally, we call Algorithm 5 for every tuple $(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r_H, r_D, r, r_F)$, $r_A, r_B, r_C, r_D, r_F, r_H \in \mathcal{R}$ and $c_v, c_w, c_x, c_y, c_z \in \mathcal{C}$. Among the non-failure solutions, we return the smallest one.

Lemma 5. *If there exists an MCSR $\mathcal{R}_T \subseteq \mathcal{R}$ involving row r (either identified to r_A or r_C) such that $\{r_A, r_B, r_C, r_H\} \subseteq \mathcal{R}_T$, $\{c_x, c_y, c_z\} \subseteq \mathcal{C}_T$ for some $\mathcal{C}_T \subseteq \mathcal{C}$, and $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{II_k})$ for some $k > 1$, then Algorithm 4 for arguments $(c_x, c_y, c_z, r_A, r_B, r_C, r_H)$ finds it.*

Lemma 6. *If there exists an MCSR $\mathcal{R}_T \subseteq \mathcal{R}$ involving row r (identified to r_E) such that $\{r_A, r_B, r_C, r_D, r_E, r_F, r_H\} \subseteq \mathcal{R}_T$, $\{c_v, c_w, c_x, c_y, c_z\} \subseteq \mathcal{C}_T$ for some $\mathcal{C}_T \subseteq \mathcal{C}$, and $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{II_k})$ for some $k > 1$, then Algorithm 5 for arguments $(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r_D, r, r_F, r_H)$ finds it.*

We now turn to evaluating the time complexity of Algorithm 5 (the time complexity of Algorithm 4 is clearly negligible with that of Algorithm 5). We first observe that, in a graph of order n, one can find a shortest uv-path that goes through a given node w in $O(n^2 \log n)$ time [26]. Indeed, it is enough to add a new vertex x, $N(x) = \{u, v\}$, and use the algorithm of Suurballe to find two vertex-disjoint paths between a source (*i.e.*, w) and a sink (*i.e.*, x) with minimum sum length. Testing emptiness of $N(r_H) \cap N(r_A) \setminus N(r_B)$, $N(r_C) \cap N(r_B) \setminus N(r_A)$, $N(r_D) \cap N(r_E) \cap N(r_F)$, and $(N(r_D) \cup N(r_E) \cup N(r_F)) \setminus (N(r_A) \cap N(r_B)))$ is a simple $O(n)$ time procedure. Cleaning any white node can be done in $O(m)$ time, and cleaning r_A and r_B in $O(n)$ time. Moreover, according to the above,

Algorithm 4. Check-$M_{II_k}(c_x, c_y, c_z, r_A, r_B, r_C, r_H)$

```
Require: A bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, four black vertices $r_A, r_B, r_C, r_H \in \mathcal{R}$
    and three white vertices $c_x, c_y, c_z \in \mathcal{C}$ such that $(r_C, c_y, r_A, c_x, r_B, c_z, r_H)$ is a path
    in $G(M)$. Row $r$ is identified either to $r_A$ or $r_C$. Furthermore, it is assumed that
    $G(M)$ does not contain any $G(M_{I_k})$ or $G(M_{II_1})$ Tucker configuration involving
    row $r$.
Ensure: Return $\mathcal{R}_T \subseteq \mathcal{R}$ such that $G(M_{II_k}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ is an
    MCSR involving row $r$, or the failure message "NO" if such a configuration does
    not exist.
 1: if $N(r_H) \cap (N(r_A) \setminus N(r_B)) \neq \emptyset$ or $N(r_C) \cap (N(r_B) \setminus N(r_A)) \neq \emptyset$ then
 2:     return "NO"
 3: end if
 4: clean$(c)$ for all $c \notin N(A) \cap N(B)$
 5: clean$(c_x, c_y, c_z)$
 6: anticlean$(r_A, r_B)$
 7: remove the vertices $r_A$ and $r_B$
 8: if there exists a $r_C r_H$-path in the pruned graph then
 9:     let $P$ be a shortest $r_C r_H$-path in the pruned graph
10:     return $\{r_A, r_B, r_C, r_H\} \cup \{r : r \in V(P) \cap \mathcal{R}\}$
11: else
12:     return "NO"
13: end if
```

finding a shortest $r_C r_H$-path that goes through r_E in the pruned graph (after having removed r_A and r_B) is a in $O((m+n)^2 \log(m+n))$ procedure. Therefore, the time complexity of one call to Algorithm 5 is $O((m+n)^2 \log(m+n))$ time.

According to Lemma 6, for a given row r, we have to call Algorithm 5 for any tuple $(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r_D, r, r_F, r_H)$, $r_A, r_B, r_C, r_D, r, r_F, r_H \in \mathcal{R}$ and $c_v, c_w, c_x, c_y, c_z \in \mathcal{C}$, and return the smallest MCSR involving row r (if such a Tucker configuration exists). There are $O(m^6 n^5)$ such tuples for a given row r, and hence trying all tuples results in a $O(m^6 < n^5 (m+n)^2 \log(m+n))$ time procedure. The exhaustive search for $G(M_{II_1})$ is a simple $O(m^4 n^4)$ time procedure. Therefore, one can find the smallest $\mathcal{R}_T \subseteq \mathcal{R}$ such that $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{II_k})$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ that is responsible for an MCSR involving row r in $O(m^6 n^5 (m+n)^2 \log(m+n))$ time (if it exists). Proposition 4 is proved.

3.4 M_{IV} and M_V Tucker Configurations

Proposition 5. *Let M be a $(0,1)$-matrix with corresponding vertex-colored bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, and r be any row of M. Finding (if it exists) a minimum cardinality $\mathcal{R}_T \subseteq \mathcal{R}$ responsible for an MCSR involving row r such that $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{IV})$ (resp. $G(M_V)$) for some $\mathcal{C}_T \subseteq \mathcal{C}$ and some $k \geq 1$ is a $O(m^3 n^6)$ (resp. $O(m^3 n^5)$) time procedure.*

Proof. The proof is by brute-force searching for a $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{IV})$ (resp. $G(M_V)$)) Tucker configuration involving row r (identified to r_A, see Fig. 1). The running time for both cases follows easily.

Algorithm 5. Check-$M_{II_k}(c_v, c_w, c_x, c_y, c_z, r_A, r_B, r_C, r_D, r_E, r_F, r_H)$

Require: A bipartite graph $G(M) = (\mathcal{R}, \mathcal{C}, E)$, seven black vertices $r_A, r_B, r_C, r_D, r_E, r_F, r_H \in \mathcal{R}$ and five white vertices $c_v, c_w, c_x, c_y, c_z \in \mathcal{C}$ such that both $(r_C, c_y, r_A, c_x, r_B, c_z, r_H)$ and $(r_D, c_v, r_E, c_w, r_F)$ are paths in $G(M)$ and $\{c_v, c_w\} \subseteq N(r_A) \cap N(r_B)$. Row r is identified to r_E. It is assumed that $G(M)$ contains neither a $G(M_{I_k})$ or a $G(M_{II_1})$ Tucker configuration involving row r.

Ensure: Return $\mathcal{R}_T \subseteq \mathcal{R}$ such that $G(M_{II_k}) = (\mathcal{R}_T, \mathcal{C}_T, E')$ for some $\mathcal{C}_T \subseteq \mathcal{C}$ is row-minimal MCSR involving row r if it exists, or the failure message "NO" is such a configuration does not exist.

```
1: if $N(r_H) \cap N(r_A) \setminus N(r_B) \neq \emptyset$ or $N(r_C) \cap N(r_B) \setminus N(r_A) \neq \emptyset$ or $N(r_D) \cap N(r_E) \cap N(r_F) \neq \emptyset$ or $(N(r_D) \cup N(r_E) \cup N(r_F)) \setminus (N(r_A) \cap N(r_B)$ then
2:     return "NO"
3: end if
4: clean$(c)$ for all $c \in N(r_E)$
5: clean$(c)$ for all $c \notin N(A) \cap N(B)$
6: clean$(c_x, c_y, c_z, c_v, c_w)$
7: anticlean$(r_A, r_B)$
8: remove the black vertices $r_A$ and $r_B$
9: if there exists a $r_C r_H$-path that goes though $r_E$ in the pruned graph then
10:     let $P$ be a shortest $r_C r_H$-path that goes though $r_E$ in the pruned graph
11:     return return $\{r_A, r_B, r_C, r_D, r_E, r_F, r_H\} \cup \{r : r \in V(P) \cap \mathcal{R}\}$
12: else
13:     return "NO"
14: end if
```

What is left is to prove that $G(M)[\mathcal{R}_T, \mathcal{C}]$ does not contain any smaller Tucker configuration. We first prove correctness for $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_{IV})$. Indeed, focus on $G(M)[\mathcal{R}_T, \mathcal{C}]$ and suppose that there exists some white vertex $c_s \in \mathcal{C} \setminus \mathcal{C}_T$ that is not a clone of some white vertex in $\mathcal{C}_T$. Then it follows that $N(c_s) = \{r_A, r_B\}$, $N(c_s) = \{r_A, r_D\}$, $N(c_s) = \{r_B, r_D\}$, or $N(c_s) = \{r_c\}$. If $N(c_s) = \{r_c)$ we are done. Otherwise, $G(M)[\mathcal{R}_T, \mathcal{C}]$ contains a (smaller) $G(M_{I_1})$ Tucker configuration. A contradiction since $G(M)$ is assumed not to contain a M_{I_k} Tucker configuration involving row r. We now turn to prove correctness for $G(M)[\mathcal{R}_T, \mathcal{C}_T] = G(M_V)$. Focus on $G(M)[\mathcal{R}_T, \mathcal{C}]$ and suppose that there exists some white vertex $c_s \in \mathcal{C} \setminus \mathcal{C}_T$ that is not a clone of some white vertex in $\mathcal{C}_T$. Then it follows that $N(c_s) = \{r_A, r_B\}$, $N(c_s) = \{r_A, r_C\}$, $N(c_s) = \{r_A, r_D\}$, or $N(c_s) = \{r_B, r_D\}$. If $N(c_s) = \{r_A, r_C\}$ we are done. Otherwise, $G(M)[\mathcal{R}_T, \mathcal{C}]$ contains a (smaller) $G(M_{I_1})$ Tucker configuration. A contradiction since $G(M)$ is assumed not to contain a M_{I_k} Tucker configuration involving row r. □

3.5 Summing Up

Table 1 summarizes our results.

Table 1.

Tucker configuration	Running time
M_{I_k}	$O(m^3 n^4)$
M_{II_k}	$O(m^6 n^5 (m+n)^2 \log(m+n))$
M_{III_k}	$O(m^5 n^5 (m+n)^2 \log(m+n))$
M_{IV}	$O(m^2 n^6)$
M_V	$O(m^3 n^5)$
Total	$O(m^6 n^5 (m+n)^2 \log(m+n))$

4 Applying Our Framework to Related Problems

Our graph pruning techniques can be used for solving related combinatorial problems. We briefly discuss these related points.

First, the property we have considered was C1P, where a matrix has C1P when the columns can be sorted in such a way that on each row the 1s are consecutive. It is simple to check that our framework can also consider the case when the property is the transpose, *i.e.*, the rows can be sorted in such a way that on each column the 1s are consecutive.

More interestingly, let us point out that our framework also implies an polynomial-time algorithm for the *Circular Ones Property* (Circ1P) studied in [10]. A matrix has the Circ1P if its columns can be ordered in such a way that all 1s or all 0s (possibly both) on each row are consecutive (it may help to consider the matrix as being wrapped around a vertical cylinder). Indeed, according to [10], Corollary 2.2, given an $m \times n$ matrix M and an arbitrary integer $1 \leq j \leq n$, one can compute a matrix M' such that M has the Circ1P if and only if M' as the C1P. Therefore, we can check in polynomial-time if a given row is involved in an MCSR for both C1P and Circ1P.

References

1. Bartholdi, J.J., Orlin, J.B., Ratliff, H.D.: Cyclic scheduling via integer programs with circular ones. Oper Res. 28(5), 1074–1085 (1980)
2. Bergeron, A., Blanchette, M., Chateau, A., Chauve, C.: Reconstructing ancestral gene orders using conserved intervals. In: Jonassen, I., Kim, J. (eds.) WABI 2004. LNCS (LNBI), vol. 3240, pp. 14–25. Springer, Heidelberg (2004)
3. Blin, G., Rizzi, R., Vialette, S.: A Faster Algorithm for Finding Minimum Tucker Submatrices. In: Ferreira, F., Löwe, B., Mayordomo, E., Mendes Gomes, L. (eds.) CiE 2010. LNCS, vol. 6158, pp. 69–77. Springer, Heidelberg (2010)
4. Booth, K.S., Lueker, G.S.: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. J. Comput. System Sci. 13, 335–379 (1976)
5. Chauve, C., Haus, U.-U., Stephen, T., You, V.P.: Minimal conflicting sets for the consecutive ones property in ancestral genome reconstruction. In: Ciccarelli, F.D., Miklós, I. (eds.) RECOMB-CG 2009. LNCS, vol. 5817, pp. 48–58. Springer, Heidelberg (2009)
6. Conforti, M., Rao, M.R.: Structural properties and decomposition of linear balanced matrices. Mathematical Programming 55, 129–168 (1992)

7. Diestel, R.: Graph Theory, 2nd edn. Graduate texts in Mathematics, vol. 173. Springer, Heidelberg (2000)
8. Dom, M.: Algorithmic aspects of the consecutive-ones property. Bull. Eur. Assoc. Theor. Comput. Sci. EATCS 98, 27–59 (2009)
9. Dom, M.: Recognition, Generation, and Application of Binary Matrices with the Consecutive-Ones Property. Dissertation. Cuvillier, Institut für Informatik, Friedrich-Schiller-Universität Jena (2009)
10. Dom, M., Guo, J., Niedermeier, R.: Approximation and fixed-parameter algorithms for consecutive ones submatrix problems. Journal of Computer and System Sciences (2009) (in Press, Corrected Proof)
11. Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pacific J. Math. 15(3), 835–855 (1965)
12. Habib, M., McConnell, R.M., Paul, C., Viennot, L.: Lex-bfs and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. Theoret. Comput. Sci. 234(12), 59–84 (2000)
13. Hassin, R., Megiddo, N.: Approximation algorithms for hitting objects with straight lines. Discrete Applied Mathematics 30(1), 29–42 (1991)
14. Hochbaum, D.S., Levin, A.: Cyclical scheduling and multi-shift scheduling: Complexity and approximation algorithms. Discrete Optimization 3(4), 327–340 (2006)
15. Hsu, W.-L.: A simple test for the consecutive ones property. J. Algorithms 43(1), 1–16 (2002)
16. Hsu, W.-L., McConnell, R.M.: Pc trees and circular-ones arrangements. Theoret. Comput. Sci. 296(1), 99–116 (2003)
17. Korte, N., Möhring, R.H.: An incremental linear-time algorithm for recognizing interval graphs. SIAM J. Comput. 18(1), 68–81 (1989)
18. Kou, L.T.: Polynomial complete consecutive information retrieval problems. SIAM J. Comput. 6(1), 67–75 (1977)
19. McConnell, R.M.: A certifying algorithm for the consecutive-ones property. In: 15th Annual ACM SIAM Symposium on Discrete Algorithms SODA 2004, pp. 768–777. ACM Press, New York (2004)
20. Mecke, S., Schöbel, A., Wagner, D.: Station location complexity and approximation. In: 5th Workshop on Algorithmic Methods and Models for Optimization of Railways ATMOS 2005, Dagstuhl, Germany (2005)
21. Mecke, S., Wagner, D.: Solving geometric covering problems by data reduction. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 760–771. Springer, Heidelberg (2004)
22. Meidanis, J., Porto, O., Telles, G.P.: On the consecutive ones property. Discrete Appl. Math. 88, 325–354 (1998)
23. Narayanaswamy, N.S., Subashini, R.: A new characterization of matrices with the consecutive ones property. Discrete Applied Mathematics 157(18), 3721–3727 (2009)
24. Ruf, N., Schöbel, A.: Set covering with almost consecutive ones property. Discrete Optimization 1(2), 215–228 (2004)
25. Stoye, J., Wittler, R.: A unified approach for reconstructing ancient gene clusters. IEEE/ACM Trans. Comput. Biol. Bioinf. 6(3), 387–400 (2009)
26. Suurballe, J.W.: Disjoint paths in networks. Networks 4, 125–145 (1974)
27. Tucker, A.C.: A structure theorem for the consecutive 1s property. Journal of Combinatorial Theory. Series B 12, 153–162 (1972)
28. Veinott, A.F., Wagner, H.M.: Optimal capacity scheduling. Oper. Res. 10, 518–547 (1962)

Two Combinatorial Criteria for BWT Images

Konstantin M. Likhomanov and Arseny M. Shur

Ural State University

Abstract. Burrows–Wheeler transform (BWT) is a block data transformation, i. e. a function on finite words. This function is used in lossless data compression algorithms and possesses interesting combinatorial properties. We study some of these properties. Namely, we prove two necessary and sufficient conditions concerning BWT images. The first one describes the words that are BWT images, while the second one explains which words can be converted to BWT images using a natural "pumping" procedure. Both conditions can be checked in linear time.

1 Introduction to BWT

In 1994, M. Burrows and D. J. Wheeler introduced a new lossless data compression scheme based on a preprocessing algorithm which is now known as Burrows–Wheeler transform (BWT). BWT maps any finite string (word) w over an ordered alphabet to its permutation $\mathsf{bwt}(w)$ defined as follows. Take all cyclic shifts of w, including w itself, sort them lexicographically and write them one under another to obtain a square table denoted by $\mathsf{T}(w)$. The word written in the last column of this table is $\mathsf{bwt}(w)$. For example, $\mathsf{bwt}(banana) = nnbaaa$ provided that $a < b < n$:

$$\mathsf{T}(banana) = \begin{bmatrix} a & b & a & n & a & \boldsymbol{n} \\ a & n & a & b & a & \boldsymbol{n} \\ a & n & a & n & a & \boldsymbol{b} \\ \mathrm{b} & \mathrm{a} & \mathrm{n} & \mathrm{a} & \mathrm{n} & \mathbf{a} \\ n & a & b & a & n & \boldsymbol{a} \\ n & a & n & a & b & \boldsymbol{a} \end{bmatrix}$$

The use of BWT in data compression is based on the *context dependence* in most data sources: almost each symbol heavily depends on preceding symbols ("left context") as well as on subsequent symbols ("right context"). For example, in an English text, only the letter 'e' (and rarely 'E') can precede the segment 'xample'. BWT groups the symbols with similar right context (the rows beginning with 'xample' end by 'e') and thus produces long runs of identical symbols. So, the output of BWT is a very good source for standard run-length or move-to-front compressing techniques. The existence of efficient algorithms to calculate both mappings bwt and bwt^{-1} allows one to build BWT-compressors for practical use. A short list of such compressors can be found in [2]. For more basics on BWT-based data compression the reader is addressed to [9]. Some of the recent studies on different aspects of BWT are gathered in [6].

Besides all possible implementations, BWT is a quite natural combinatorial mapping possessing nice properties. The results obtained in this direction mainly concerns words having "simple" BWT images [7,8,10,11] and permutations associated with BWT (see, e.g., [4,5]). In this paper, we study the following natural question: *are there any words whose BWT image has a given prescribed form*?

We consider two versions of this question. The first version is the simplest one: decide, whether a given word is a BWT image. E. g., is there a word w such that $\mathsf{bwt}(w) = banana$? For the second version, consider the representation of any word u in the form $b_1^{i_1} \ldots b_k^{i_k}$, where $b_1, \ldots, b_k$ are letters, $b_j \neq b_{j+1}$ for all $j < k$, and $i_1, \ldots, i_k \in \mathbb{N}$. We say that the word $b_1 \ldots b_k$ is the *block sequence* of u. Note that if u is a BWT image of a context dependent fragment of data, then the block sequence of u is usually much shorter than u itself. So, the question is whether there exists a BWT image with a given block sequence. E. g., is there a word w such that $\mathsf{bwt}(w) = b^{i_1}a^{i_2}n^{i_3}a^{i_4}n^{i_5}a^{i_6}$ for some $i_1, \ldots, i_6 \in \mathbb{N}$? Note that the papers [8,10,11] were devoted to the study of the words over the alphabet $\{a_1 < \ldots < a_n\}$ whose BWT images have the block sequence $a_n \ldots a_1$.

In the paper, we answer the above question, providing efficiently testable necessary and sufficient conditions for both mentioned versions of this question. After necessary preliminaries, in Sect. 3 we characterize the property "to be a BWT image" in terms of permutations and give a linear-time algorithm to check the characterizing condition. This algorithm can be also used for efficient calculation of BWT preimages. In Sect. 4, we find a combinatorial property (of words) which is equivalent to the property "to be a block sequence of a BWT image". This property can be checked in linear time also. In addition, we show that a block sequence of a BWT image is a block sequence of infinitely many BWT images and provide a quadratic-time algorithm that builds BWT images from a block sequence.

2 Definitions and Preliminaries

Let $\Sigma = \{a_1, a_2, \ldots, a_n\}$ be a fixed ordered alphabet with $a_1 < a_2 < \ldots < a_n$. The induced lexicographic ordering on the set Σ^* of all finite words over Σ will be also denoted by $<$. Unless explicitly specified, we consider the words over Σ. A word u is *primitive* if the equality $u = w^k$ implies $k = 1$. The length of a word u is denoted by $|u|$. Let r be the right cyclic shift operation on Σ^*: $\mathsf{r}(uc) = cu$ for any $u \in \Sigma^*$, $c \in \Sigma$. Words u and v are *conjugates* if $u = \mathsf{r}^i(v)$ for some i. A word u has $|u|$ distinct conjugates if and only if u is primitive.

Take a word u and let $m = |u|$. The $m \times m$ *BWT table* $\mathsf{T}(u)$ and the BWT image $\mathsf{bwt}(u)$ are defined above.

As usual, S_m denotes the set of all permutations of the set $\{1, \ldots, m\}$. Let $\theta \in S_m$, $i \in \{1, \ldots, m\}$. The set $\{i, \theta(i), \theta^2(i), \theta^3(i), \ldots\}$ is an *orbit* of θ. The orbits of θ form a partition of the set $\{1, \ldots, m\}$. For an m-element sequence $y = (y_1, \ldots, y_m)$ of any origin (e.g. for a word), the *action of θ on y* is defined by $\theta(y_1, \ldots, y_m) = (y_{\theta(1)}, \ldots, y_{\theta(m)})$.

Let $w = c_1 \ldots c_m$ be a word. There is a unique permutation $\sigma_w \in S_m$ such that (1) if $c_i < c_j$, then $\sigma_w(i) < \sigma_w(j)$ and (2) if $c_i = c_j$, then the difference

$\sigma_w(i) - \sigma_w(j)$ has the same sign as $i - j$. This permutation is said to be the *stable sorting* of w. The difference $\sigma_w(i) - i$ will be called the *shift* of the i-th letter of w under the stable sorting and denoted by $s_w(i)$. Thus, $\sigma_w(i) = i + s_w(i)$.

Every word w has a unique representation $w = c_1^{p_1} \ldots c_l^{p_l}$ such that $c_i \neq c_{i+1}$ for all $i < l$. The words $c_i^{p_i}$ are *blocks* of w. It is easy to see that the stable sorting of w moves each block as a whole, so that the shifts of all letters in a block coincide. Then we can define the *shift of a block* under the stable sorting as the shift of its letters. We write $\mathsf{Sb}_w(t)$ for the shift of the block $c_t^{p_t}$. To calculate this shift, look at the blocks preceding $c_t^{p_t}$ in the word w and in the "stably sorted" word $\sigma_w(w)$. If $c_i = c_t$, then the stable sorting preserves the mutual location of the blocks $c_i^{p_i}$ and $c_t^{p_t}$. Let $c_i \neq c_t$. If $i < t$, then $c_i^{p_i}$ is on the left of the block $c_t^{p_t}$ in the word w; the stable sorting moves $c_i^{p_i}$ to the right of $c_t^{p_t}$ if and only if $c_i > c_t$. Similarly, the stable sorting moves the block $c_j^{p_j}$ from the right to the left of $c_t^{p_t}$ if and only if $j > t$ and $c_j < c_t$. Since $|c_i^{p_i}| = p_i$, we get

$$\mathsf{Sb}_w(t) = \sum_{\substack{i<t,\\ c_i>c_t}} p_i - \sum_{\substack{j>t,\\ c_j<c_t}} p_j \qquad (1)$$

3 BWT and Structure of Permutations

Proposition 1. *Let u be a primitive word. Then $\sigma_{\mathsf{bwt}(u)}$ has a single orbit.*

Proof. The rows of the table $\mathsf{T}(u)$ represent the conjugates of u and hence are all different. Let us move the rightmost column of $\mathsf{T}(u)$ to the leftmost position, obtaining the table T'. In this way, each row v of $\mathsf{T}(u)$ will be transformed to $\mathsf{r}(v)$, so the set of rows will not change. Thus, if we write the rows of T' in lexicographic order, we will get the table $\mathsf{T}(u)$ once again. This transformation is the action of some permutation on the sequence of rows of the table T'. Let us denote this permutation by θ. Since the first column of T' contains the word $w = \mathsf{bwt}(u)$, we see that $\theta(w)$ is the sequence of all letters of u (as well as of w) written in lexicographical order.

Let us show that θ is the stable sorting of w. Indeed, for any $v, v' \in \Sigma^*$ and $c \in \Sigma$, the condition $|v| = |v'|$ implies the equivalence of the relations $cv < cv'$, $v < v'$, and $vc < v'c$. Take the numbers $i < j$ such that $w_i = w_j = c$. Let $v_i c$ and $v_j c$ be the i-th and the j-th rows of $\mathsf{T}(u)$, respectively. Then $v_i c < v_j c$, and hence $cv_i < cv_j$. The word cv_i $[cv_j]$ is the ith [respectively, the jth] row of the table T', and hence, the $\theta(i)$th [respectively, $\theta(j)$th] row of $\mathsf{T}(u)$. Thus, if $i < j$ and $w_i = w_j$, then $\theta(i) < \theta(j)$. So, $\theta = \sigma_w$ by definition.

Let the words $u, \mathsf{r}(u), \ldots, \mathsf{r}^{|u|-1}(u)$ occupy the rows $i_0, \ldots, i_{|u|-1}$ of $\mathsf{T}(u)$, respectively. By definition of θ, we have

$$\theta(i_0) = i_1,\ \theta(i_1) = i_2, \ldots,\ \theta(i_{|u|-1}) = i_0.$$

So, $\{i_0, \ldots, i_{|u|-1}\}$ is an orbit of θ. Since all conjugates of u are different, this orbit contains $|u|$ elements and hence is the only orbit of θ. □

Proposition 2. *Let* $w = c_1^{dp_1} c_2^{dp_2} \dots c_l^{dp_l}$, $\bar{w} = c_1^{p_1} \dots c_l^{p_l}$. *Then the orbits of* σ_w *are exactly the sets* $\{di_1{-}r, di_2{-}r, \dots, di_k{-}r\}$ *such that* $\{i_1, i_2, \dots, i_k\}$ *is an orbit of* $\sigma_{\bar{w}}$ *and* $0 \le r < d$. *In particular,* d *divides the number of orbits of* σ_w.

Proof. It follows from (1) that d divides $\mathsf{Sb}_w(t)$ for all $t = 1, \dots, l$, that is, divides $s_w(i)$ for all $i = 1, \dots, |w|$. Hence, $\sigma_w(i) = i + s_w(i) \equiv_k i$ and the set $A = \{d, 2d, \dots, |w| = |\bar{w}|d\}$ is invariant under σ_w. Then the restriction $\sigma_w|_A$ of σ_w to A is a permutation on A. Being the stable sorting of w, σ_w induces the stable sorting of the subsequence formed by the dth, $(2d)$th, ..., $(|\bar{w}|d)$th letters of w. This subsequence is equal to $\bar{w}$, so $\sigma_w|_A$ is isomorphic to $\sigma_{\bar{w}}$. Hence, for each orbit $\{i_1, i_2, \dots, i_k\}$ of $\sigma_{\bar{w}}$ there is an orbit $\{di_1, di_2, \dots, di_k\}$ of σ_w. Now consider an arbitrary number between 1 and $|w|$. It can be uniquely represented as $dj{-}r$ with $0 \le r < d$. Since d divides the lengths of all blocks of w, the $(dj{-}r)$th and (dj)th letters of w belong to the same block, whence $s_w(dj{-}r) = s_w(dj)$. But then $\sigma_w(dj{-}r) = dj - r + s_w(dj{-}r) = dj + s_w(dj) - r = \sigma_w(dj) - r$. So, if dj belongs to the orbit $(di_1, di_2, \dots, di_k)$, then $dj{-}r$ belongs to the orbit $(di_1{-}r, di_2{-}r, \dots, di_k{-}r)$. Thus, the permutation σ_w indeed has all the required orbits. Since these orbits exhaust the set $\{1, \dots, |w|\}$, the proof is complete. □

Corollary 1. *If* u *is a primitive word, then the lengths of blocks in the word* $\mathsf{bwt}(u)$ *are relatively prime.*

The following statement is a stronger version of a well-known property of BWT [7].

Proposition 3. *For a word* u, $\mathsf{bwt}(u) = c_1^{kp_1} \dots c_l^{kp_l}$ *if and only if there exists a word* v *such that* $u = v^k$ *and* $\mathsf{bwt}(v) = c_1^{p_1} \dots c_l^{p_l}$.

Proof. Sufficiency trivially follows from the fact that if $u = v^k$, then there are k instances of the row $\bar{v}^k$ in the table $\mathsf{T}(u)$ for each row $\bar{v}$ of $\mathsf{T}(v)$.

Now prove necessity. Let $\bar{u}$ be the primitive word such that $u = \bar{u}^t$. Then $\mathsf{bwt}(\bar{u}) = c_1^{\bar{p}_1} \dots c_l^{\bar{p}_l}$, where $t\bar{p}_i = kp_i$ (compare to the previous paragraph). The numbers $\bar{p}_i$ are relatively prime by Corollary 1. Hence, t is the greatest common divisor of the lengths of blocks in u. Therefore, $t = k\bar{t}$ for some $\bar{t}$. Then $p_i = \bar{t}\bar{p}_i$ and the word $v = \bar{u}^{\bar{t}}$ satisfies the required conditions. □

Now we can prove the main theorem of this section.

Theorem 1. *A word* w *is a BWT image if and only if the number of orbits of* σ_w *equals the greatest common divisor of the lengths of blocks in* w.

Proof. To prove necessity, suppose that $w = \mathsf{bwt}(u^d)$, where u is a primitive word and $d \ge 1$. Let $\bar{w} = \mathsf{bwt}(u) = c_1^{p_1} \dots c_l^{p_l}$. By Proposition 3, $w = c_1^{dp_1} \dots c_l^{dp_l}$. Then σ_w has d orbits by Proposition 2, because $\sigma_{\bar{w}}$ has a single orbit by Proposition 1. By Corollary 1, the numbers $p_1, \dots, p_l$ are relatively prime, so d is the greatest common divisor of the lengths of blocks in w.

Now let us prove sufficiency. Let d be the greatest common divisor of the lengths of blocks in w, $m = |w|$. First consider the case $d = 1$. We create a table T

of size $m \times m$, ith column of which contains the word $\sigma_w^i(w)$ for any $i = 1, \ldots, |w|$ (note that $\sigma_w^m(w) = w$). The word in the ith row of T will be denoted by u_i. The action of σ_w on the sequence of rows of T can be considered as the action of σ_w on each column of T. The resulting table $\sigma_w(T)$ will contain in its columns, starting from the left, the words $\sigma_w^2(w), \ldots, \sigma_w^m(w), \sigma_w(w)$, respectively. Hence, if we move the rightmost column of $\sigma_w(T)$ to the leftmost position, we will get the table T again. During this transformation from T back to T, we first move the word u_1 from row 1 to the row $\sigma_w(1)$ and then replace u_1 by $\mathsf{r}(u_1)$. So, $u_{\sigma_w(1)} = \mathsf{r}(u_1)$. Similarly, the word $u_{\sigma_w(1)}$ is moved to the row $\sigma_w^2(1)$ and then replaced by its right shift, and so on. Thus, $u_{\sigma_w^i(1)} = \mathsf{r}^i(u_1)$ for all i. Since σ_w has a single orbit, we have $\{1, \sigma_w(1), \sigma_w^2(1), \ldots, \sigma_w^m(1)\} = \{1, 2, \ldots, m\}$, and then the set of rows of T coincides with the set of cyclic shifts of the word u_1.

Now let us prove that the rows of T are lexicographically ordered. Suppose they are not. Then $i < j$ and $u_i > u_j$ for some i, j. Among all such pairs (i, j), choose the one for which the longest common prefix of u_i and u_j has minimal length. The letters in the first column of T are lexicographically ordered by definition of σ_w, so u_i and u_j must begin with the same letter. Let $u_i = cv_i$, $u_j = cv_j$, $\sigma_w^{-1}(i) = i'$, $\sigma_w^{-1}(j) = j'$. In the previous paragraph we learned that in this case $u_i = \mathsf{r}(u_{i'})$, $u_j = \mathsf{r}(u_{j'})$. So, $u_{i'} = v_i c$, $u_{j'} = v_j c$. Then $u_i > u_j$ implies that $u_{i'} > u_{j'}$. Since the words $u_{i'}$ and $u_{j'}$ have shorter common prefix than u_i and u_j, we have $i' > j'$. But both the i'th and the j'th letters of w are equal to c, and the inequality $\sigma_w(i') = i < j = \sigma_w(j')$ contradicts to the definition of stable sorting. Thus, we have proved that the rows of T are lexicographically ordered cyclic shifts of the word u_1. Since the last column of T contains the word w, we have $w = \mathsf{bwt}(u_1)$ by definition.

Now let $d > 1$. Denote the word formed by the dth, $(2d)$th, ..., mth letters of w by $\bar{w}$. It follows from Proposition 2 that the number of orbits of the permutation σ_w is d times the number of orbits of $\sigma_{\bar{w}}$. Hence, $\sigma_{\bar{w}}$ has a single orbit, and $\bar{w} = \mathsf{bwt}(u)$ for some u, as we have already proved. Then $w = \mathsf{bwt}(u^d)$ by Proposition 3. The theorem is proved. □

Corollary 2. *If w is a BWT image having at least two different letters, then σ_w has no fixed points.*

Following [3], we call a permutation $\theta \in S_m$ *indecomposable* if no set $\{1, \ldots, k\}$ with $k < m$ is invariant under θ.

Corollary 3. *If w is a BWT image having at least two different letters, then σ_w is indecomposable.*

Proof. Note that a set invariant under θ is a union of orbits of θ. Let d be the greatest common divisor of the lengths of blocks in w and suppose that the set $\{1, \ldots, k\}$ is invariant under σ_w. By Theorem 1 and Proposition 2, the numbers 1 and $|w| - d + 1$ belong to the same orbit of σ_w, yielding $k \geq |w| - d + 1$. But some other orbit contains both d and $|w|$. Since $d < |w| - d + 1$, we get $k = |w|$. □

The criterion given in Theorem 1 can be checked in $O(|w|)$ time. To do this, create a list for each letter of Σ and process w in one pass, recording each letter's position in the corresponding list. For example, if $w = nnbaaa$, then the lists look as follows: $a : [4, 5, 6]$, $b : [3]$, $n : [1, 2]$. Concatenating these lists according to the order on Σ, one recovers the permutation σ_w^{-1} which has the same orbits as σ_w. In our example, $\sigma_w^{-1} = \left(\begin{smallmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 4 & 5 & 6 & 3 & 1 & 2 \end{smallmatrix}\right)$. While traversing w, we can also compute the greatest common divisor d of the lengths of blocks (by keeping the length of the current block and the greatest common divisor of the lengths of all previous blocks). In view of Proposition 2, it remains to choose an arbitrary integer $j \in [1, |w|]$ and check whether the orbit of σ_w^{-1} containing j is of length $\frac{|w|}{d}$. For the word $w = nnbaaa$, it is the case, while for $w = banana$ the permutation σ_w^{-1} has the orbits $\{1, 2, 4\}$ and $\{3, 5, 6\}$, thus indicating that *banana* is not a BWT image.

Remark 1. If a word w is a BWT image, the algorithm described in the previous paragraph allows one to restore any prescribed row of the corresponding BWT table T. Indeed, ith column of T contains the word $\sigma^i(w)$ (see the proof of Theorem 1). Hence, the ith letter of some jth row of T is the $\sigma_w^{-i}(j)$th letter of w. When calculating the length of the orbit of σ_w^{-1} containing the number j, we compute the numbers $\sigma_w^{-1}(j)$, $\sigma_w^{-2}(j)$, etc. Writing down the letters in the corresponding positions of the word w, we obtain a prefix u_1 of the jth row of T such that $|u_1| = \frac{|w|}{d}$ and $w = \mathsf{bwt}(u_1^d)$. Thus, having a word which is supposed to be a BWT image, one can do two things simultaneously: check the correctness of the word and find any of its preimages.

Remark 2. The use of lists in the considered algorithm can be avoided. Namely, if we scan w in advance, counting the occurrences of each letter, we can directly record the permutation σ_w^{-1} in a $|w|$-element array. This requires storing the numbers up to $|w|$, so formally the algorithm uses $O(|w| \log |w|)$ additional space. However, assuming we use one-byte letters, the required space is $3|w|$ if the size of processed input word does not exceed 16 Mb, while $4|w|$ space allows to process the words up to 4 Gb.

4 Block Structure of BWT Images

In this section we describe a transformation of words which preserves the property "to be a BWT image". This transformation is then used to characterize possible block sequences of BWT images and to build an algorithm constructing BWT images with a given block sequence.

Let us introduce the concept of *generalized blocks*. Let $w = c_1^{p_1} \ldots c_l^{p_l}$ for some $c_i \in \Sigma$, $p_i \geq 0$ (note that the letters c_i and c_{i+1} may coincide and p_i may be equal to zero). The words $c_i^{p_i}$ are *generalized blocks* of w, and the word $c_1 \ldots c_l$ is a *generalized block sequence* of w. The above factorization of w is not unique, so when we need to fix it, we use the notation $[w] = [c_1^{p_1}, c_2^{p_2}, \ldots, c_l^{p_l}]$. If the factorization is fixed, we can define block shifts under the stable sorting using the same formula (1) as for usual blocks. This allows us to define shifts for generalized blocks of zero length. For example, the stable sorting of $w = ab^6a^2$ maps $[w] = [a^1, b^3, a^0, b^1, b^2, a^2]$ to $[a^1, a^0, a^2, b^3, b^1, b^2]$, so we have $\mathsf{Sb}_{[w]}(3) = -3$.

Our main idea is to change the length of a generalized block by a multiple of its shift.

Proposition 4. *Suppose that* $[w] = [c_1^{p_1}, c_2^{p_2}, \ldots, c_t^{p_t}, \ldots, c_l^{p_l}]$ *and* $k \in \mathbb{Z}$*. Then* $\gcd(p_1, \ldots, p_t, \ldots, p_l) = \gcd(p_1, \ldots, p_t + k\mathsf{Sb}_{[w]}(t), \ldots, p_l)$*.*

Proof. Let $d = \gcd(p_1, \ldots, p_t, \ldots, p_l)$, $d' = \gcd(p_1, \ldots, p_t + k\mathsf{Sb}_{[w]}(t), \ldots, p_l)$, and $d_1 = \gcd(p_1, \ldots, p_{t-1}, p_{t+1}, \ldots, p_l)$. Since the formula for $\mathsf{Sb}_{[w]}(t)$ does not include p_t, the number d_1 divides $\mathsf{Sb}_{[w]}(t)$. Then $d = \gcd(p_t, d_1) = \gcd(p_t + k\mathsf{Sb}_{[w]}(t), d_1) = d'$. □

Proposition 5. *Suppose that* $[w] = [c_1^{p_1}, \ldots, c_t^{p_t}, \ldots, c_l^{p_l}]$*, an integer* k *satisfies the inequality* $p_t + k\mathsf{Sb}_{[w]}(t) \geq 0$*, and* $[w]' = [c_1^{p_1}, \ldots, c_t^{p_t + k\mathsf{Sb}_{[w]}(t)}, \ldots, c_l^{p_l}]$*. Then the permutations* σ_w *and* $\sigma_{w'}$ *have the same number of orbits.*

Proof. The case $k\mathsf{Sb}_{[w]}(t) = 0$ is trivial. Note that it is sufficient to prove the proposition for the case $k = \operatorname{sgn}(\mathsf{Sb}_{[w]}(t))$. Indeed, $\mathsf{Sb}_{[w']}(t) = \mathsf{Sb}_{[w]}(t)$, so the statement can be proved by induction for every k having the same sign as $\mathsf{Sb}_{[w]}(t)$. Changing the roles of w and w', we similarly operate with the case $k\mathsf{Sb}_{[w]}(t) < 0$.

Suppose $\mathsf{Sb}_{[w]}(t) > 0$ (and $k = 1$). Let $P = p_1 + p_2 + \ldots + p_t$ and define an injective function f as follows:

$$f(i) = \begin{cases} i, & \text{if } 1 \leq i \leq P; \\ i + \mathsf{Sb}_{[w]}(t), & \text{if } P < i \leq |w|. \end{cases}$$

Note that if the ith letter of w is the kth letter of the jth block, then the $f(i)$th letter of w' is also the kth letter of the jth block.

Let $1 \leq i \leq |w|$ and let the ith letter of w belong to the jth block. Then $\sigma_w(i) = i + s_w(i) = i + \mathsf{Sb}_{[w]}(j)$ and similarly $\sigma_{w'}(f(i)) = f(i) + \mathsf{Sb}_{[w']}(j)$. We aim to prove that if the permutation σ_w maps i to i', then the numbers $f(i)$ and $f(i')$ belong to the same orbit of $\sigma_{w'}$. Six cases are to be considered.

Case 1: $j \leq t$, $\sigma_w(i) \leq P$. Then either $j = t$ or the jth block of w precedes the tth one both before and after sorting. Either way, the expression (1) for $\mathsf{Sb}_{[w]}(j)$ does not include p_t, so we have $\mathsf{Sb}_{[w']}(j) = \mathsf{Sb}_{[w]}(j)$ and

$$\sigma_{w'}(f(i)) = f(i) + \mathsf{Sb}_{[w']}(j) = i + \mathsf{Sb}_{[w]}(j) = \sigma_w(i) = f(\sigma_w(i)).$$

Case 2: $j \leq t$, $\sigma_w(i) > P + \mathsf{Sb}_{[w]}(t)$. Then the jth block of w precedes the tth one before sorting, but not after it. Hence, p_t appears in the expression (1) for $\mathsf{Sb}_{[w]}(j)$ with the $+$ sign and $\mathsf{Sb}_{[w']}(j) = \mathsf{Sb}_{[w]}(j) + \mathsf{Sb}_{[w]}(t)$. So,

$$\sigma_{w'}(f(i)) = f(i) + \mathsf{Sb}_{[w']}(j) = i + \mathsf{Sb}_{[w]}(j) + \mathsf{Sb}_{[w]}(t) = \sigma_w(i) + \mathsf{Sb}_{[w]}(t) = f(\sigma_w(i)).$$

Case 3: $j > t$, $\sigma_w(i) \leq P$. Here the jth block of w precedes the tth one after sorting, but not before it. Then p_t is included in the expression (1) for $\mathsf{Sb}_{[w]}(j)$ with the $-$ sign and we have $\mathsf{Sb}_{[w']}(j) = \mathsf{Sb}_{[w]}(j) - \mathsf{Sb}_{[w]}(t)$. Hence,

$$\sigma_{w'}(f(i)) = f(i) + \mathsf{Sb}_{[w']}(j) = i + \mathsf{Sb}_{[w]}(t) + \mathsf{Sb}_{[w]}(j) - \mathsf{Sb}_{[w]}(t) = \sigma_w(i) = f(\sigma_w(i)).$$

Case 4: $j > t$, $\sigma_w(i) > P + \mathsf{Sb}_{[w]}(t)$. Since the jth block of w follows the tth block both before and after sorting, p_t does not appear in the expression (1) for $\mathsf{Sb}_{[w]}(j)$, so $\mathsf{Sb}_{[w']}(j) = \mathsf{Sb}_{[w]}(j)$. Then we have

$$\sigma_{w'}(f(i)) = f(i)+\mathsf{Sb}_{[w']}(j) = i+\mathsf{Sb}_{[w]}(t)+\mathsf{Sb}_{[w]}(j) = \sigma_w(i)+\mathsf{Sb}_{[w]}(t) = f(\sigma_w(i)).$$

Case 5: $j \le t$, $P < \sigma_w(i) \le P+\mathsf{Sb}_{[w]}(t)$. As in case 1, $\mathsf{Sb}_{[w']}(j) = \mathsf{Sb}_{[w]}(j)$. Note that $\sigma_w(i)$th letter of w' belongs to its tth block, so we have

$$\sigma^2_{w'}(f(i)) = \sigma_{w'}(f(i) + \mathsf{Sb}_{[w']}(j)) = \sigma_{w'}(i + \mathsf{Sb}_{[w]}(j)) = \\ \sigma_{w'}(\sigma_w(i)) = \sigma_w(i) + \mathsf{Sb}_{[w']}(t) = \sigma_w(i) + \mathsf{Sb}_{[w]}(t) = f(\sigma_w(i)).$$

Case 6: $j > t$, $P < \sigma_w(i) \le P + \mathsf{Sb}_{[w]}(t)$. As in case 3, $\mathsf{Sb}_{[w']}(j) = \mathsf{Sb}_{[w]}(j) - \mathsf{Sb}_{[w]}(t)$. Since $\sigma_w(i)$th letter of w' belongs to its tth block, we have

$$\sigma^2_{w'}(f(i)) = \sigma_{w'}(f(i) + \mathsf{Sb}_{[w']}(j)) = \sigma_{w'}(i + \mathsf{Sb}_{[w]}(t) + \mathsf{Sb}_{[w]}(j) - \mathsf{Sb}_{[w]}(t)) = \\ \sigma_{w'}(\sigma_w(i)) = \sigma_w(i) + \mathsf{Sb}_{[w']}(t) = \sigma_w(i) + \mathsf{Sb}_{[w]}(t) = f(\sigma_w(i)).$$

Thus, if some set $\{i_1, i_2, \ldots, i_r\}$ is an orbit of σ_w, then the numbers $f(i_1)$, $f(i_2), \ldots, f(i_r)$ with possible inclusion of some numbers from the set $\{P+1, \ldots, P+\mathsf{Sb}_{[w]}(t)\}$ form an orbit of $\sigma_{w'}$. No orbit can be fully contained in the set of positions of the tth block of w', because all these positions have the shift $\mathsf{Sb}_{[w]}(t) > 0$. Hence there is a bijection between the sets of orbits of permutations σ_w and $\sigma_{w'}$.

Now let $\mathsf{Sb}_{[w]}(t)$ be negative. Then $k = -1$ and $|w'| = |w| - \mathsf{Sb}_{[w]}(t)$. Consider the ordered alphabet $\overleftarrow{\Sigma}$ obtained from Σ by reversing the order. Suppose that the words $v, v' \in \overleftarrow{\Sigma}$ are given by their generalized block sequences $[v] = [c_l^{p_l}, \ldots, c_t^{p_t}, \ldots, c_1^{p_1}]$ and $[v'] = [c_l^{p_l}, \ldots, c_t^{p_t - \mathsf{Sb}_{[w]}(t)}, \ldots, c_1^{p_1}]$. It is easy to see that the action of the stable sorting σ_v on $(|w|, \ldots, 1)$ is the same as the action of σ_w on $(1, \ldots, |w|)$ and the action of $\sigma_{v'}$ on $(|w'|, \ldots, 1)$ is the same as the action of $\sigma_{w'}$ on $(1, \ldots, |w'|)$. Also, $\mathsf{Sb}_{[v]}(l-t+1) = -\mathsf{Sb}_{[w]}(t)$. Hence, σ_v and $\sigma_{v'}$ have the same number of orbits by the above argument for positive shifts, and so do σ_w and $\sigma_{w'}$. $\square$

Combining Theorem 1, Proposition 4, and Proposition 5, we get

Corollary 4. *Under the conditions of Proposition 5, if w is a BWT image, then so is w'.*

Before formulating the main result of this section, we need to introduce and study one more notion. We say that a word w has a *global ascent* if it can be factorized as $w = uv$, where u, v are nonempty and the maximal letter of u is less than or equal to the minimal letter of v.

Proposition 6. *If w is a BWT image having at least two different letters, then it has no global ascents.*

Proof. If $w = uv$ is a decomposition corresponding to a global ascent, then the set $\{1, \dots, |u|\}$ is invariant under σ_w by definition of stable sorting. This contradicts to Corollary 3. □

Proposition 7. *If a nonempty word has no global ascents, then we can delete one of its letters so that the remaining word will have no global ascents as well.*

Proof. Suppose that w have no global ascents. The case $|w| \leq 2$ is trivial, so suppose $|w| \geq 3$. Let c and b be the maximal and second maximal letters of w, respectively. Also, let i (respectively, j) be the rightmost position of w in which the letter c (respectively, b) occurs. Assume that if we delete c in ith position, we will get a word with a global accent (otherwise we are done with the proof). Then $w = uv$, where after the removal of c both words u and v remain nonempty and the maximal letter of u is no greater than the minimal letter of v. So, the removal of c changes either the maximal letter of u or the minimal letter of v. The latter possibility obviously cannot take place; hence, the ith position lies inside u and contains the only occurrence of c in w.

Now consider the jth position. If $j < i$, then u contains b's while v does not. This contradicts to the assumption that u and v form a global ascent after removing the letter c. Thus, $j > i$. It remains to check that the word obtained from w by deleting b in jth position has no global ascents. Let $w = xy$ be an arbitrary factorization of w into two nonempty words. If the ith position is in x, then the maximal letter of x is greater than any letter of y, and this situation would not change if we delete the letter b somewhere. If the ith position is in y, then the minimal letter of y is less than b, because otherwise w has a global ascent. Hence, the deletion of b in y does not change the minimal letter of y and then does not produce a global ascent. Therefore, the deletion of b in jth position results in a word with no global ascents. □

We are approaching the main result of this section, which characterizes the words $v = c_1 \dots c_l$ that can be transformed to BWT images by a sort of "pumping": each letter c_i can be replaced by $c_i^{p_i}$ for an arbitrary positive number p_i. In particular, we characterize the words that are block sequences of BWT images.

Theorem 2. *Let $v = c_1 \dots c_l$ be an arbitrary word with at least two different letters. A BWT image of the form $c_1^{p_1} \dots c_l^{p_l}$, where $p_i > 0$ for all i, exists if and only if v has no global ascents.*

Proof. If v has a global ascent, then any word of the form $c_1^{p_1} \dots c_l^{p_l}$ also has a global ascent. By Proposition 6, such a word is not a BWT image.

We prove the converse statement by induction on l. The inductive base is trivial: if $l = 2$, then $c_1 > c_2$ and $\mathsf{bwt}(v) = v = c_1c_2$. For the inductive step, let $l > 2$. By Proposition 7, there exists a number $t \in \{1, \dots, l\}$ such that the word $c_1 \dots c_{t-1}c_{t+1} \dots c_l$ has no global ascents. By the inductive assumption, there exists a BWT image u which can be factorized in generalized blocks as $[u] = [c_1^{p_1}, \dots, c_{t-1}^{p_{t-1}}, c_{t+1}^{p_{t+1}}, \dots, c_l^{p_l}]$. Consider one more factorization of u, namely, $[u]' = [c_1^{p_1}, \dots, c_{t-1}^{p_{t-1}}, c_t^0, c_{t+1}^{p_{t+1}}, \dots, c_l^{p_l}]$. If $|\mathsf{Sb}_{[u]'}(t)| = k > 0$, then by Corollary 4 the factorization $[c_1^{p_1}, \dots, c_{t-1}^{p_{t-1}}, c_t^k, c_{t+1}^{p_{t+1}}, \dots, c_l^{p_l}]$ defines a BWT image.

Now let $\mathsf{Sb}_{[u]'}(t) = 0$. Consider the formula (1) for $\mathsf{Sb}_{[u]'}(t)$. If the right-hand expression is empty, then $c_t \geq c_i$ for $i < t$ and $c_t \leq c_i$ for $i > t$. But then v obviously has a global ascent. Hence the formula includes p_j for some $j > t$. Since the block $c_j^{p_j}$ of $[u]'$ is nonempty, we have $|\mathsf{Sb}_{[u]'}(j)| = q > 0$ by Corollary 2. We replace the block $c_j^{p_j}$ in $[u]'$ by $c_j^{p_j+q}$, obtaining a factorization $[\bar{u}]$ of a new word $\bar{u}$. The word $\bar{u}$ is a BWT image by Corollary 4 and clearly $|\mathsf{Sb}_{[\bar{u}]}(t)| = q$. So, applying Corollary 4 once again we deduce that the word $[c_1^{p_1}, \ldots, c_{t-1}^{p_{t-1}}, c_t^q, c_{t+1}^{p_{t+1}}, \ldots, c_j^{p_j+q}, \ldots, c_l^{p_l}]$ is a BWT image. The inductive step is proved. □

Remark 3. The check of v for global ascents takes $O(l)$ time. One needs to traverse v one time in each direction to calculate the maximum letters of all prefixes and the minimum letters of all suffixes, respectively. E. g., the word *banana* has no global ascents and then is a block sequence of some BWT image.

The proof of Theorem 2 gives us the following algorithm to "pump" a word to a BWT image. Note that Corollary 4 ensures that the obtained image can be "pumped" further to get infinitely many BWT images.

Algorithm. (After some steps, comments are given.)
Input: a word $v = c_1 \ldots c_l$ with at least two different letters.
Output: a BWT image of the form $c_1^{p_1} \ldots c_l^{p_l}$ with $p_1, \ldots, p_l > 0$, or "NO".

1. Check whether v has a global ascent. If it does, return "NO" and stop. (Theorem 2)
2. Construct a sequence of words without global ascents $v_1 = v, v_2, \ldots, v_{l-1}$, each being obtained by deleting one letter from the previous one. (Proposition 7)
3. Let $[w_{l-1}] = [c_1^1, c_2^1]$, where $v_{l-1} = c_1c_2$; let $j = l-1$.
4. If $j = 1$, return w_1 and stop. (An exit point of the main loop)
5. Let $v_{j-1} = c_1 \ldots c_{l-j+2}$. Word v_j was obtained from v_{j-1} by deleting some letter, say, c_k, the factorization $[w_j] = [c_1^{p_1}, \ldots, c_{k-1}^{p_{k-1}}, c_{k+1}^{p_{k+1}}, \ldots, c_{l-j+2}^{p_{1-j+2}}]$ of the word w_j is already built. Let $[w_j]' = [c_1^{p_1}, \ldots, c_{k-1}^{p_{k-1}}, c_k^0, c_{k+1}^{p_{k+1}}, \ldots, c_{l-j+2}^{p_{1-j+2}}]$. Let $q = \mathsf{Sb}_{[w_j]'}(k)$.
6. If $q \neq 0$, then let $[w_{j-1}] = [c_1^{p_1}, \ldots, c_{k-1}^{p_{k-1}}, c_k^q, c_{k+1}^{p_{k+1}}, \ldots, c_{l-j+2}^{p_{1-j+2}}]$, decrease j by 1 and go to step 4.
7. We come here if $q = 0$. Find any i such that the formula (1) for $\mathsf{Sb}_{[w]'}(k)$ includes p_i; w. l. o. g., $i < k$. Let $r = |\mathsf{Sb}_{[w]'}(i)|$, $[w_{j-1}] = [c_1^{p_1}, \ldots, c_i^{p_i+r}, \ldots, c_{k-1}^{p_{k-1}}, c_k^r, c_{k+1}^{p_{k+1}}, \ldots, c_{l-j+2}^{p_{1-j+2}}]$, decrease j by 1 and go to step 4.

Example. In order to demonstrate all steps of the above algorithm, we construct a BWT image from the word *dacbcda*. The word has no global ascents, so this is possible.

By deleting letters in the way, described in the proof of Proposition 7, we get $v_1 = v = dacbcda, v_2 = dacbca, v_3 = dacba, v_4 = daba, v_5 = daa, v_6 = da$.

Then we put $[w_6] = [d^1, a^1]$ and enter the main cycle to obtain, subsequently,
$[w_6]' = [d^1, a^1, a^0]$, $\mathsf{Sb}_{[w_6]'}(3) = -1$, $[w_5] = [d^1, a^1, a^1]$;
$[w_5]' = [d^1, a^1, b^0, a^1]$ and $\mathsf{Sb}_{[w_5]'}(3) = p_4 - p_1 = 0$; take $i = 1$; $\mathsf{Sb}_{[w_5]'}(1) = 2$, $[w_4] = [d^3, a^1, b^2, a^1]$;
$[w_4]' = [d^3, a^1, c^0, b^2, a^1]$, $\mathsf{Sb}_{[w_4]'}(3) = p_4 + p_5 - p_1 = 0$; take $i = 4$; $\mathsf{Sb}_{[w_4]'}(4) = -2$, $[w_3] = [d^3, a^1, c^2, b^4, a^1]$;
$[w_3]' = [d^3, a^1, c^2, b^4, c^0, a^1]$, $\mathsf{Sb}_{[w_3]'}(5) = -2$, $[w_2] = [d^3, a^1, c^2, b^4, c^2, a^1]$;
$[w_2]' = [d^3, a^1, c^2, b^4, c^2, d^0, a^1]$, $\mathsf{Sb}_{[w]'_1}(6) = 1$, and finally,
$[w_1] = [d^3, a^1, c^2, b^4, c^2, d^1, a^1]$.
The obtained word is indeed a BWT image:
$dddaccbbbbccda = \mathsf{bwt}(abcbcdaddbcbcd)$.

Remark 4. Let us estimate the time complexity of the above algorithm. Rather than actually constructing the word sequence $\{v_1, \dots, v_{l-1}\}$ we can just enumerate the letters of v in the order of their removal. From the proof of Proposition 7 it follows that to determine the next letter in this order we need to find two greatest letters among the remaining ones and check one word for global ascents. This takes linear time (see Remark 3), so the whole enumeration can be performed in $O(l^2)$ time. After that, we set lengths of the two remaining blocks to 1 and all other lengths of blocks — to 0. Then on each step we compute the shift of the current block ($O(l)$ time); if it is equal to zero, we find another block whose length we need to change ($O(l)$ time) and compute its shift (also $O(l)$ time). This procedure will be repeated $l-2$ times, so our algorithm has quadratic time complexity overall.

Let us note that if some of the words v_i obtained on step 2 is a BWT image, we can suppose $w_i = v_i$ and immediately jump to step 4. If we perform a linear-time check for BWT image on each of v_i, this won't affect the worst-case asymptotics, but will improve the best-case complexity: the algorithm will run in linear time if v itself is a BWT image.

References

1. Burrows, M., Wheeler, D.J.: A Block-sorting lossless data compression algorithm. SRC Research Report 124, Digital Systems Research Center, Palo Alto (1994)
2. BWT compression comparison, `http://compressionratings.com/bwt.html`
3. Comtet, L.: Advanced combinatorics. Reidel, Dordrecht (1974)
4. Crochemore, M., Désarménien, J., Perrin, D.: A note on the Burrows–Wheeler transformation. Theor. Comput. Sci. 332(1-3), 567–572 (2005)
5. Duval, J.-P., Lefebvre, A.: Words over ordered alphabet and suffix permutations. RAIRO Theor. Inform. Appl. 36, 249–259 (2002)
6. Ferragina, P., Manzini, G., Muthukrishnan, S. (eds.): The Burrows–Wheeler transform: special issue of Theor. Comput. Sci., vol. 387(3) (2007)
7. Mantaci, S., Restivo, A., Sciortino, M.: Burrows–Wheeler transform and Sturmian words. Inf. Process. Lett. 86, 241–246 (2003)
8. Mantaci, S., Restivo, A., Sciortino, M.: Combinatorial aspects of the Burrows-Wheeler transform. In: Proc. WORDS 2003, vol. 27, pp. 292–297. TUCS Gen. Publ. (2003)

9. Manzini, J.: An analysis of the Burrows–Wheeler transform. J. ACM 48(3), 207–230 (2001)
10. Restivo, A., Rosone, G.: Balanced Words Having Simple Burrows-Wheeler Transform. In: Diekert, V., Nowotka, D. (eds.) DLT 2009. LNCS, vol. 5583, pp. 431–442. Springer, Heidelberg (2009)
11. Simpson, J., Puglisi, S.J.: Words with simple Burrows–Wheeler transforms. Electronic J. Comb. 15, #R83 (2008)

Recent Results on Polynomial Identity Testing

Amir Shpilka

Faculty of Computer Science
Technion - Israel Institute of Technology
Haifa, Israel
shpilka@cs.technion.com
http://www.cs.technion.ac.il/~shpilka/

Polynomial Identity Testing (PIT) is a fundamental problem in algebraic complexity: We are given a circuit computing a multivariate polynomial, over some field $\mathbb{F}$, and we have to determine whether it is identically zero or not. Note that we want the polynomial to be identically zero and not just to be equal to the zero function so, for example, $x^2 - x$ is the zero function over $\mathbb{F}_2$ but not the zero polynomial. The importance of this problem follows from its many applications: Algorithms for primality testing [2, 3], for deciding if a graph contains a perfect matching [19, 20, 8] and more, are based on reductions to the PIT problem (see the introduction of [18] for more applications).

There are two well studied scenarios in which the PIT problem is considered. The first is the so called *black-box* model in which the circuit is given as a black-box and we can access it only by querying its value on inputs of our choice. It is clear that every such algorithm must produce a test set for the circuit. Namely, a set of points such that if the circuit vanishes on all the points in the set then the circuit computes the zero polynomial. Another well studied scenario is the non black-box case in which the circuit is given to us as input. In particular, we have access to the polynomials that are being computed at various gates of the circuit. Clearly this is an 'easier' version of the problem, yet PIT is extremely difficult in this model as well.

Determining the complexity of PIT is one of the greatest challenges of theoretical computer science. It is one of a few problems for which we have coRP algorithms but no sub-exponential time deterministic algorithms. Indeed, many clever randomized algorithms are known for the general PIT question [26, 29, 10, 9, 18, 2] whereas sub-exponential time deterministic algorithms are known only for very restricted models. One explanation for this state of affairs is the strong relation between PIT and lower bounds for arithmetic circuits. Although seemingly very different, the problem of derandomizing PIT (i.e., that of giving efficient deterministic algorithms for the problem) is closely related to the problem of proving super-polynomial lower bounds for arithmetic circuits. In [14] Kabanets and Impagliazzo showed that efficient deterministic algorithms for PIT imply that NEXP does not have polynomial size arithmetic circuits. Specifically, if PIT can be solved deterministically in polynomial time, even in the non black-box model, then either the Permanent cannot be computed by polynomial size arithmetic circuits or NEXP $\not\subseteq$ P/poly. That is, we get a super polynomial lower bound either for NEXP or for the Permanent. In [14] it was also shown that from super-polynomial lower bounds for arithmetic circuits one can design a deterministic quasi-polynomial time algorithm for PIT. In [12] (almost) analogous results for bounded depth circuits were obtained. In [13, 1] it was observed that polynomial time derandomization of PIT, in the black-box model, implies exponential lower bounds for arithmetic circuits. These results show the strong connection between PIT and lower bounds and indicate how difficult and important this problem is.

Because of the strong connection to proving lower bounds it is not surprising that the PIT problem becomes very interesting already for bounded depth circuits. Specifically, [4] proved that polynomial time derandomization of PIT for depth 4 circuits already implies exponential lower bounds for *general* arithmetic circuits. In combination with the results of [14] this gives a quasi-polynomial time derandomization of PIT for general arithmetic circuits. Hence, the problem of derandomizing PIT for depth 4 circuits is as difficult (and as important) as the problem for general arithmetic circuits.

Currently, deterministic subexponential PIT algorithms are known for non-commutative arithmetic formulae [21], for depth 3 circuits with a small top fan-in [11, 17, 15, 16, 24], for read-k formulas [5] and for multilinear depth 4 circuits with bounded top fan-in (as well as several other restricted versions of depth 4 circuits [6, 23, 27, 25]). It is not known whether derandomizing PIT for depth 4 multilinear circuit implies a derandomization of PIT for general multilinear circuits. However, such a derandomization does imply an exponential lower bound for general multilinear circuits, thus improving the slightly super linear bound of [22].

Another line of research concerning PIT focused on better understanding the relation to other computational problems. In [28] a relation between PIT and multivariate polynomial factorization was found. Specifically, [28] showed that one can derandomize PIT if and only if one can derandomize the problem of computing variable disjoint factors of a given multilinear polynomial (that relation holds both in the black-box and non black-box models). In [7] a relation between deterministic PIT to circuit lower bounds and the isolation lemma was found.

In this talk we shall survey some of the results on PIT that were mentioned above and give a list of what we think are the most accessible and important open problems.

References

1. Agrawal, M.: Proving lower bounds via pseudo-random generators. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 92–105. Springer, Heidelberg (2005)
2. Agrawal, M., Biswas, S.: Primality and identity testing via chinese remaindering. J. ACM 50(4), 429–443 (2003)
3. Agrawal, M., Kayal, N., Saxena, N.: Primes is in P. Annals of Mathematics 160(2), 781–793 (2004)

4. Agrawal, M., Vinay, V.: Arithmetic circuits: A chasm at depth four. In: Proceedings of the 49th Annual FOCS, pp. 67–75 (2008)
5. Anderson, M., van Melkebeek, D., Volkovich, I.: Derandomizing Polynomial Identity Testing for Multilinear Constant-Read Formulae. In: Electronic Colloquium on Computational Complexity (ECCC), vol. 17 (2010)
6. Arvind, V., Mukhopadhyay, P.: The monomial ideal membership problem and polynomial identity testing. In: Tokuyama, T. (ed.) ISAAC 2007. LNCS, vol. 4835, pp. 800–811. Springer, Heidelberg (2007)
7. Arvind, V., Mukhopadhyay, P.: Derandomizing the isolation lemma and lower bounds for circuit size. In: Goel, A., Jansen, K., Rolim, J.D.P., Rubinfeld, R. (eds.) APPROX and RANDOM 2008. LNCS, vol. 5171, pp. 276–289. Springer, Heidelberg (2008)
8. Chari, S., Rohatgi, P., Srinivasan, A.: Randomness-optimal unique element isolation with applications to perfect matching and related problems. SIAM J. on Computing 24(5), 1036–1050 (1995)
9. Chen, Z., Kao, M.: Reducing randomness via irrational numbers. SIAM J. on Computing 29(4), 1247–1256 (2000)
10. DeMillo, R.A., Lipton, R.J.: A probabilistic remark on algebraic program testing. Inf. Process. Lett. 7(4), 193–195 (1978)
11. Dvir, Z., Shpilka, A.: Locally decodable codes with 2 queries and polynomial identity testing for depth 3 circuits. SIAM J. on Computing 36(5), 1404–1434 (2006)
12. Dvir, Z., Shpilka, A., Yehudayoff, A.: Hardness-randomness tradeoffs for bounded depth arithmetic circuits. SIAM J. on Computing 39(4), 1279–1293 (2009)
13. Heintz, J., Schnorr, C.P.: Testing polynomials which are easy to compute (extended abstract). In: Proceedings of the 12th annual STOC, pp. 262–272 (1980)
14. Kabanets, V., Impagliazzo, R.: Derandomizing polynomial identity tests means proving circuit lower bounds. Computational Complexity 13(1-2), 1–46 (2004)
15. Karnin, Z.S., Shpilka, A.: Deterministic black box polynomial identity testing of depth-3 arithmetic circuits with bounded top fan-in. In: Proceedings of the 23rd Annual CCC, pp. 280–291 (2008)
16. Kayal, N., Saraf, S.: Blackbox polynomial identity testing for depth 3 circuits. In: Electronic Colloquium on Computational Complexity (ECCC), vol. 32 (2009)
17. Kayal, N., Saxena, N.: Polynomial identity testing for depth 3 circuits. Computational Complexity 16(2), 115–138 (2007)
18. Lewin, D., Vadhan, S.: Checking polynomial identities over any field: Towards a derandomization? In: Proceedings of the 30th Annual STOC, pp. 428–437 (1998)
19. Lovasz, L.: On determinants, matchings, and random algorithms. In: Budach, L. (ed.) Fundamentals of Computing Theory. Akademia-Verlag (1979)
20. Mulmuley, K., Vazirani, U., Vazirani, V.: Matching is as easy as matrix inversion. Combinatorica 7(1), 105–113 (1987)
21. Raz, R., Shpilka, A.: Deterministic polynomial identity testing in non commutative models. Computational Complexity 14(1), 1–19 (2005)
22. Raz, R., Shpilka, A., Yehudayoff, A.: A lower bound for the size of syntactically multilinear arithmetic circuits. SIAM J. on Computing 38(4), 1624–1647 (2008)
23. Saxena, N.: Diagonal circuit identity testing and lower bounds. In: Aceto, L., Damgård, I., Goldberg, L.A., Halldórsson, M.M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) ICALP 2008, Part I. LNCS, vol. 5125, pp. 60–71. Springer, Heidelberg (2008)
24. Saxena, N., Seshadhri, C.: Blackbox identity testing for bounded top fanin depth-3 circuits: the field doesn't matter. In: Electronic Colloquium on Computational Complexity (ECCC), vol. 17 (2010)

25. Saraf, S., Volkovich, I.: Black-Box Identity Testing of Depth-4 Multilinear Circuits. In: Proceedings of the 43rd Annual STOC (2011)
26. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. JACM 27(4), 701–717 (1980)
27. Shpilka, A., Volkovich, I.: Improved polynomial identity testing for read-once formulas. In: APPROX-RANDOM, pp. 700–713 (2009)
28. Shpilka, A., Volkovich, I.: On the relation between polynomial identity testing and finding variable disjoint factors (2009) (submitted)
29. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: Symbolic and Algebraic Computation, pp. 216–226 (1979)

Towards Approximate Matching in Compressed Strings: Local Subsequence Recognition[*]

Alexander Tiskin

Department of Computer Science, University of Warwick,
Coventry CV4 7AL, UK

Abstract. A grammar-compressed (GC) string is a string generated by a context-free grammar. This compression model includes LZ78 and LZW compression as a special case. We consider the longest common subsequence problem and the local subsequence recognition problem on a GC-text against a plain pattern. We show that, surprisingly, both problems can be solved in time that is within a polylogarithmic factor of the best existing algorithms for the same problems on a plain text. In a wider context presented elsewhere, we use these results as a stepping stone to efficient approximate matching on a GC-text.

1 Introduction

String compression is a classical area of computer science. It is natural to ask whether compressed strings can be processed efficiently without decompression. Early examples of such algorithms were given e.g. by Amir et al. [1] and by Rytter [13].

We consider the following general model of compression.

Definition 1. *Let t be a string of length n (typically large). String t will be called a* grammar-compressed string (GC-string)*, if it is generated by a context-free grammar, also called a* straight-line program (SLP)*. An SLP of length $\bar{n}$, $\bar{n} \leq n$, is a sequence of $\bar{n}$* statements*. A statement numbered k, $1 \leq k \leq \bar{n}$, has one of the following forms: $t_k = \alpha$, where α is an alphabet character, or $t_k = t_i t_j$, where $1 \leq i, j < k$.*

We identify every symbol t_r with the string it expands to; in particular, we have $t = t_{\bar{n}}$. In general, the plain string length n can be exponential in the GC-string length $\bar{n}$.

Grammar compression includes as a special case the classical LZ78 and LZW compression schemes by Ziv, Lempel and Welch [22,20]. It should also be noted that certain other compression methods, such as e.g. LZ77 and run-length compression, do not fit directly into the grammar compression model.

[*] Research supported by the Centre for Discrete Mathematics and Its Applications (DIMAP), University of Warwick, and by the Royal Society Leverhulme Trust Senior Research Fellowship.

The algorithms in this paper will take as input a grammar-compressed *text string* t of length n, generated by an SLP of length $\bar{n}$, and a plain *pattern string* p of length m. We aim at algorithms with running time that is a low-degree polynomial in m, $\bar{n}$, but is independent of n (which could be exponential in $\bar{n}$).

In this paper, we consider the longest common subsequence (LCS) problem, and the local subsequence recognition problem on a GC-text against a plain pattern. We show that, surprisingly, both problems can be solved in time $O(m \log m \cdot \bar{n})$, which is within a polylogarithmic factor of the best existing algorithms for the same problems on a plain text against a plain pattern. In a wider context presented in [14], we use these results as a stepping stone to efficient approximate matching on a GC-text.

2 General Techniques

In this section and the next, we recall the algorithmic framework of semi-local string comparison, developed in [15,16,18], and fully presented in [14].

2.1 Preliminaries

For indices, we will use either integers, or *odd half-integers*:

$$\{\ldots, -2, -1, 0, 1, 2, \ldots\} \qquad \left\{\ldots, -\tfrac{5}{2}, -\tfrac{3}{2}, -\tfrac{1}{2}, \tfrac{1}{2}, \tfrac{3}{2}, \tfrac{5}{2}, \ldots\right\}$$

For ease of reading, odd half-integer variables will be indicated by hats (e.g. $\hat{\imath}$, $\hat{\jmath}$). Ordinary variable names (e.g. i, j, with possible subscripts or superscripts), will normally denote integer variables, but can sometimes denote a variable that may be either integer, or odd half-integer. We denote integer and odd half-integer *intervals* by

$$[i : j] = \{i, i+1, \ldots, j-1, j\} \qquad \langle i : j \rangle = \left\{i + \tfrac{1}{2}, i + \tfrac{3}{2}, \ldots, j - \tfrac{3}{2}, j - \tfrac{1}{2}\right\}$$

When dealing with pairs of numbers, we will often use geometric language and call them *points*. We will write $(i_0, j_0) \ll (i_1, j_1)$ if $i_0 < i_1$, $j_0 < j_1$, and $(i_0, j_0) \lessgtr (i_1, j_1)$ if $i_0 < i_1$, $j_0 > j_1$. We will call these strict partial orders $\ll$- and $\lessgtr$-*dominance*. When visualising points, we will use the matrix indexing convention: the first coordinate in a pair increases downwards, and the second coordinate rightwards.

We use standard terminology for geometric dominance and other partial orders. In particular, a set of elements forms a *chain*, if they are pairwise comparable, and an *antichain*, if they pairwise incomparable. Note that a $\ll$-chain is a $\lessgtr$-antichain, and vice versa. An element in a partially ordered set is *minimal* (respectively, *maximal*), if, in terms of the partial order, it does not dominate (respectively, is not dominated by) any other element in the set. All minimal (respectively, maximal) elements in a partially ordered set form an antichain.

Given two index ranges I, J, it will be convenient to denote their Cartesian product by $(I \mid J)$. We extend this notation to Cartesian products of intervals:

$$[i_0 : i_1 \mid j_0 : j_1] = ([i_0 : i_1] \mid [j_0 : j_1])$$

$$\langle i_0 : i_1 \mid j_0 : j_1 \rangle = (\langle i_0 : i_1 \rangle \mid \langle j_0 : j_1 \rangle)$$

Given index ranges I, J, a *vector over* I is indexed by $i \in I$, and a *matrix over* $(I \mid J)$ is indexed by $i \in I$, $j \in J$. We will denote by A^T the transpose of matrix A.

Definition 2. *Let D be a matrix over $\langle i_0 : i_1 \mid j_0 : j_1 \rangle$. Its* distribution matrix *D^Σ over $[i_0 : i_1 \mid j_0 : j_1]$ is defined by*

$$D^\Sigma(i, j) = \textstyle\sum_{\hat{\imath} \in \langle i : i_1 \rangle, \hat{\jmath} \in \langle j_0 : j \rangle} D(\hat{\imath}, \hat{\jmath})$$

for all $i \in [i_0 : i_1]$, $j \in [j_0 : j_1]$.

Definition 3. *A* permutation *(respectively,* subpermutation*)* matrix *is a zero-one matrix containing exactly one (respectively, at most one) nonzero in every row and every column.*

Typically, (sub)permutation matrices will be indexed by odd half-integers. When dealing with such matrices, we will write "nonzeros" for "index pairs corresponding to nonzeros", as long as this does not lead to confusion. We will normally assume that a (sub)permutation matrix with n nonzeros is given implicitly by a compact data structure of size $O(n)$, that allows constant-time access to each nonzero both by the row and by the column index.

2.2 Semi-local LCS

We will consider strings of characters taken from an alphabet. Two alphabet characters α, β *match*, if $\alpha = \beta$, and *mismatch* otherwise. In addition to alphabet characters, we introduce a special non-alphabet *wildcard character* '$\diamond$', which matches itself and all other characters.

It will be convenient to index strings by odd half-integer, rather than integer indices, e.g. string $a = \alpha_{\frac{1}{2}} \alpha_{\frac{3}{2}} \ldots \alpha_{m-\frac{1}{2}}$. We will index strings as vectors, writing e.g. $a(\hat{\imath}) = \alpha_{\hat{\imath}}$, $a\langle i : j \rangle = \alpha_{i+\frac{1}{2}} \ldots \alpha_{j-\frac{1}{2}}$. String concatenation will be denoted by juxtaposition.

Given a string, we distinguish between its contiguous *substrings*, and not necessarily contiguous *subsequences*. Special cases of a substring are *a prefix* and *a suffix* of a string. Unless indicated otherwise, our algorithms will take as input a string a of length m, and a string b of length n.

Definition 4. *Given strings a, b, the* longest common subsequence (LCS) problem *asks for the length of the longest string that is a subsequence of both a and b. We will call this length the* LCS score *of strings a, b.*

A classical solution to the global LCS problem is given by the dynamic programming algorithm [12,19], which runs in time $O(mn)$. The best known algorithms for the LCS problem [10,4,2] improve this running time by (model-dependent) polylogarithmic factors.

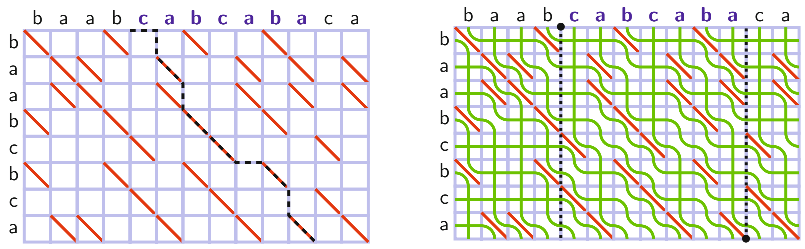


Fig. 1. Alignment dag $G_{a,b}$, a highest-scoring path, nonzeros of $P_{a,b}$ as seaweeds

Definition 5. *Given strings a, b, the* semi-local LCS problem *asks for the LCS scores as follows: a against every substring of b (the* string-substring LCS scores*); every prefix of a against every suffix of b (the* prefix-suffix LCS scores*); symmetrically, the* substring-string LCS scores *and the* suffix-prefix LCS scores*, defined as above but with the roles of a and b exchanged.*

It turns out that, although more general than the LCS problem, the semi-local LCS problem can also be solved within (model-dependent) polylogarithmic factors of $O(mn)$.

A special case of the semi-local LCS problem is the *local subsequence recognition problem*, which, given a text t and a pattern p, asks for the substrings in t containing p as a subsequence. This problem can also be regarded as a basic form of approximate pattern matching.

In certain contexts, such as when one of the input strings (say, a) is very long, we may not wish to deal with all the substrings of the longer string, but still to consider the other three components of the semi-local LCS problem.

Definition 6. *Given strings a, b, the* three-way semi-local LCS problem *asks for the prefix-suffix, suffix-prefix and substring-string LCS scores, but excludes the string-substring LCS scores.*

2.3 Alignment Dags and Score Matrices

It is well-known that an instance of the LCS problem can be represented by a dag (directed acyclic graph) on a rectangular grid of nodes, where character matches correspond to edges scoring 1, and mismatches to edges scoring 0.

Definition 7. *An* alignment dag *is a weighted dag, defined on the set of nodes $v_{l,i}$, $l \in [l_0 : l_1]$, $i \in [i_0, i_1]$. The edge and path weights are called* scores*. For all $l \in [l_0 : l_1]$, $\hat{l} \in \langle l_0 : l_1 \rangle$, $i \in [i_0, i_1]$, $\hat{\imath} \in \langle i_0 : i_1 \rangle$, the alignment dag contains: horizontal edge $v_{l,\hat{\imath}-\frac{1}{2}} \to v_{l,\hat{\imath}+\frac{1}{2}}$ and vertical edge $v_{\hat{l}-\frac{1}{2},i} \to v_{\hat{l}+\frac{1}{2},i}$, both with score* 0*; diagonal edge $v_{\hat{l}-\frac{1}{2},\hat{\imath}-\frac{1}{2}} \to v_{\hat{l}+\frac{1}{2},\hat{\imath}+\frac{1}{2}}$ with score either* 0 *or* 1*.*

An alignment dag can be viewed as an $(l_1 - l_0) \times (i_1 - i_0)$ grid of *cells*. An instance of the semi-local LCS problem on strings a, b corresponds to an $m \times n$ alignment dag $G_{a,b}$; a cell indexed by $\hat{l} \in \langle 0 : m \rangle$, $\hat{\imath} \in \langle 0 : n \rangle$ is called a *match cell*, if $a(\hat{l})$ matches $b(\hat{\imath})$, and a *mismatch cell* otherwise (recall that the strings may contain wildcard characters). The diagonal edges in match cells have score 1, and in mismatch cells score 0. Clearly, the diagonal edges with score 0 do not affect maximum node-to-node scores, and can therefore be ignored.

Example 1. Figure 1 shows the alignment dag for strings $a =$ "baabcbca", $b =$ "baabcabcabaca". All edges are directed left-to-right and top-to-bottom; the blue (respectively, red) colour[1] corresponds to edge weight 0 (respectively, 1). The highlighted path of score 5 corresponds to the string-substring LCS score for string a against substring $b\langle 4 : 11\rangle =$ "cabcaba".

The semi-local LCS problem is equivalent to the problem of finding the highest-scoring paths for each of the four possible path types (top-to-bottom, left-to-bottom, top-to-right, and left-to-right), and every conforming pair of endpoints on the boundary of the alignment dag. The analysis of different path types can be simplified by padding one of the input strings with wildcard characters. Accordingly, we need to consider an extended alignment dag for string a over $\langle 0 : m\rangle$ against string $\diamond^m b \diamond^m$ over $\langle -m : m+n\rangle$.

Definition 8. *Given strings a, b, the corresponding* semi-local score matrix *is a matrix over $[-m : n \mid 0 : m+n]$, defined by $H_{a,b}(i,j) = \max score(v_{0,i} \rightsquigarrow v_{m,j})$, where $i \in [-m : n]$, $j \in [0 : m+n]$, and the maximum is taken across all paths between the given endpoints $v_{0,i}$, $v_{m,j}$ in the $m \times (2m+n)$ alignment dag $G_{a,\diamond^m b \diamond^m}$. If $i = j$, we have $H_{a,b}(i,j) = 0$. By convention, if $j < i$, then we let $H_{a,b}(i,j) = j - i < 0$.*

Example 2. Figure 2 shows the matrix $H_{a,b}$, giving all semi-local LCS scores for strings a, b as in the previous examples. The entry $H_{a,b}(4,11) = 5$ is circled.

The solution for each of the four components of the semi-local LCS problem can be obtained from the semi-local score matrix $H_{a,b}$ by simple linear transformations; see [14] for details.

Theorem 1. *We have*

$$H_{a,b}(i,j) = j - i - P^{\Sigma}_{a,b}(i,j) = m - P^{T\Sigma T}_{a,b}(i,j)$$

where $P_{a,b}$ is a permutation matrix over $\langle -m : n \mid 0 : m+n\rangle$. In particular, string a is a subsequence of substring $b\langle i : j\rangle$ for some $i, j \in [0 : n]$, if and only if $P^{T\Sigma T}_{a,b}(i,j) = 0$. (Note that the expression $P^{T\Sigma T}_{a,b}$ involves two matrix transpositions.)

Proof. The proof is based on a careful analysis of the properties of semi-local score matrices. See [14, Section 3.2] for details. □

The key idea of our approach is to view Theorem 1 as a description of an implicit solution to the semi-local LCS problem. The semi-local score matrix $H_{a,b}$ is represented implicitly by the nonzeros of the permutation matrix $P_{a,b}$.

[1] For colour illustrations, the reader is referred to the online version of this paper.

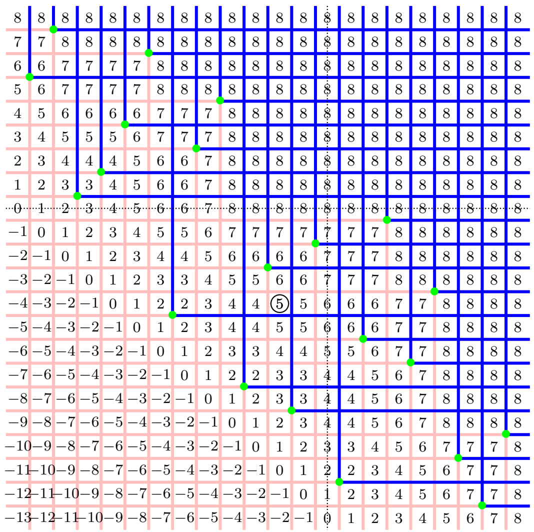


Fig. 2. Matrices $H_{a,b}$ and $P_{a,b}$

Definition 9. *Given strings a, b, the* semi-local seaweed matrix *is a permutation matrix $P_{a,b}$ over $\langle -m : n \mid 0 : m+n \rangle$, defined by Theorem 1.*

Example 3. Figure 2 shows the unit-anti-Monge property of matrix $H_{a,b}$ by a coloured grid pattern, where the red (respectively, blue) lines separate matrix elements that differ by 1 (respectively, by 0). The nonzeros of the semi-local seaweed matrix $P_{a,b}$ over $\langle -8 : 13 \mid 0 : 8+13 \rangle$ are shown by green bullets.

The nonzeros of $P_{a,b}$ that are $\lessgtr$-dominated by the point $(4, 11)$ correspond to the green bullets lying below and to the left of the circled entry. Note that there are exactly two such nonzeros, therefore $P^{\Sigma}_{a,b}(4, 11) = 2$, and that $H_{a,b}(4, 11) = 11 - 4 - P^{\Sigma}_{a,b}(4, 11) = 11 - 4 - 2 = 5$.

The nonzeros of $P_{a,b}$ that $\lessgtr$-dominate the point $(4, 11)$ correspond to the green bullets lying above and to the right of the circled entry. Note that there are exactly three such nonzeros, therefore $P^{T\Sigma T}_{a,b}(4, 11) = 3$, and that $H_{a,b}(4, 11) = 8 - P^{T\Sigma T}_{a,b}(4, 11) = 8 - 3 = 5$.

Example 4. Figure 1 shows matrix $P_{a,b}$ as a *seaweed braid*, laid out directly on the alignment dag $G_{a,b}$. The nonzeros correspond to seaweeds, laid out as paths in the dual graph. We say that a seaweed goes *from $\hat{\imath}$ to $\hat{\jmath}$*, if it originates between the nodes $v_{0,\hat{\imath}-\frac{1}{2}}$ and $v_{0,\hat{\imath}+\frac{1}{2}}$, and terminates between the nodes $v_{8,\hat{\jmath}-\frac{1}{2}}$ and $v_{8,\hat{\jmath}+\frac{1}{2}}$. In particular, every nonzero $P_{a,b}(\hat{\imath}, \hat{\jmath}) = 1$, where $\hat{\imath}, \hat{\jmath} \in \langle 0 : 13 \rangle$, is represented by a seaweed going from $\hat{\imath}$ to $\hat{\jmath}$. The remaining seaweeds, originating or terminating at the sides of the dag, correspond to nonzeros $P_{a,b}(\hat{\imath}, \hat{\jmath}) = 1$, where either $\hat{\imath} \in \langle -8 : 0 \rangle$ or $\hat{\jmath} \in \langle 13 : 8+13 \rangle$ (or both). For the purposes of this example, the specific layout of the seaweeds between their endpoints is not important.

The full set of $8+13 = 21$ nonzeros corresponds to the full set of 21 seaweeds in Figure 1. The two nonzeros that are $\lessgtr$-dominated by the point $(4, 11)$ correspond to the two seaweeds (from 4.5 to 6.5 and from 7.5 to 9.5) fitting completely between the two dashed vertical lines $i = 4$ and $j = 11$. The three nonzeros that $\lessgtr$-dominate the point $(4, 11)$ correspond to the three seaweeds (from 0.5 to the right boundary; from 1.5 to 13.5; from 3.5 to the right boundary) piercing both these vertical lines.

With minimal modification, the definition of seaweed matrix can also be applied separately to each component of the semi-local LCS problem.

Definition 10. *Given strings a, b, the corresponding* string-substring, prefix-suffix, suffix-prefix and substring-string seaweed matrices *are respectively the following subpermutation submatrices of the semi-local seaweed matrix $P_{a,b}$:*

$$P^{▖}_{a,b} = P_{a,b}\langle 0 : n \mid 0 : n\rangle \qquad P^{▗}_{a,b} = P_{a,b}\langle 0 : n \mid n : m+n\rangle$$
$$P^{▘}_{a,b} = P_{a,b}\langle -m : 0 \mid 0 : n\rangle \qquad P^{▝}_{a,b} = P_{a,b}\langle -m : 0 \mid n : m+n\rangle$$

Example 5. Figure 2 shows the partition of $P_{a,b}$ in Definition 10 by thin dotted lines. The string-substring, prefix-suffix, suffix-prefix and substring-string submatrices are respectively on the bottom-left, bottom-right, top-left and top-right. Note that the substring-string submatrix $P^{▝}_{a,b}$ is trivial, with all the entries equal to 0; this is due to the fact that the whole string a is a subsequence of b.

The nonzeros of each matrix introduced in Definition 10 can be regarded as an implicit solution to the corresponding component of the semi-local LCS problem. Similarly, the last three matrices taken together can serve as an implicit solution to the three-way semi-local LCS problem.

2.4 Score Matrix Composition

We now describe how the previously introduced techniques can be applied within a divide-and-conquer framework.

Let b', b'' be nonempty strings of length n', n'' respectively. Given the concatenation string $b = b'b''$ of length $n = n' + n''$, a substring $b\langle i' : i''\rangle$ with $i' \in [0 : n' - 1]$, $i'' \in [n' + 1 : n]$ will be called a *cross-substring*. In other words, a cross-substring of b is a concatenation of a nonempty suffix of b' and a nonempty prefix of b''.

Definition 11. *Given strings a and $b = b'b''$, the corresponding* seaweed cross-matrix *is the subpermutation matrix $P^{■}_{a,b',b''} = P_{a,b}\langle -m : n' \mid n' : m + n\rangle$.*

In contrast with the seaweed matrices in Definition 10, the dimensions of the seaweed cross-matrices introduced by Definition 11 depend on the lengths of individual component strings in the concatenation. Hence, the notation for these matrices involves three, rather than two, string subscripts.

Theorem 2. *Given the nonzeros of matrices $P^{▗}_{a,b'}$, $P^{▘}_{a,b'}$, $P^{▝}_{a,b'}$, $P^{▗}_{a,b''}$, $P^{▘}_{a,b''}$, $P^{▝}_{a,b''}$, it is possible to compute the nonzeros of matrices $P^{▗}_{a,b}$, $P^{▘}_{a,b}$, $P^{▝}_{a,b}$, as well as $P^{▙}_{a,b',b''}$, in time $O\big(m \log \min(m, n', n'')\big)$.*

Proof. The algorithm is based on a recent result [18] that allows fast distance multiplication of unit-Monge matrices (i.e. distribution matrices of permutation metrices). See [14, Section 3.3] for details. □

3 Subsequences in Compressed Strings

3.1 Three-Way Semi-local LCS

Previous work. We recall that the LCS problem on a pair of plain strings can be solved within a (model-dependent) polylogarithmic factor of $O(mn)$. The LCS problem on a pair of GC-strings has been considered by Lifshits and Lohrey [9], and proven to be NP-hard.

Recall that we aim at algorithms on a GC-text against a plain pattern, with running time independent of n (which could be exponential in $\bar{n}$). This rules out any attempt at solving the full semi-local LCS problem, since the resulting semi-local seaweed matrix would require memory $O(m+n)$. However, we are still able to consider the three-way semi-local LCS problem, excluding the computation of LCS on substrings of t.

A GC-text is a special case of a context-free language, which consists of a single string. Therefore, the LCS problem between a GC-text and a plain pattern can be regarded as a special case of the edit distance problem between a context-free language given by a grammar of size $\bar{n}$, and a pattern string of size m. For this more general problem, Myers [11] gave an algorithm running in time $O(m^3\bar{n} + m^2 \cdot \bar{n} \log \bar{n})$. In [17], we gave an algorithm for the three-way semi-local LCS problem between a GC-text and a plain pattern, running in time $O(m^{1.5}\bar{n})$. Lifshits [8] asked whether the LCS problem in the same setting can be solved in time $O(m\bar{n})$.

New result. A new algorithm for the three-way semi-local LCS problem, running in time $O(m \log m \cdot \bar{n})$, can be obtained by an application of the techniques described in Section 2. The resulting algorithm improves on existing algorithms in running time, and approaches an answer to Lifshits' question within a logarithmic factor.

Algorithm 1 (Three-way semi-local LCS).

Input: SLP of length $\bar{n}$, generating text t of length n; plain pattern p of length m.

Output: nonzeros of matrices $P^{▗}_{p,t}$, $P^{▘}_{p,t}$, $P^{▝}_{p,t}$.

Description. First, we observe that, although the output matrices contain at most m nonzeros, the index range of these nonzeros is of size $m+n$, which may be exponentially larger. To avoid an exponential growth of the indices, we will clean up the range by removing unused indices, and deleting the corresponding zero row-column pairs from the matrices. Formally, we describe this process as an order-preserving remapping of the index range.

First phase. Recursion on the input SLP generating t.

Recursion base: $n = \bar{n} = 1$. The output can be computed by a linear sweep of string p.

Recursive step: $n \geq \bar{n} > 1$. Let $t = t't''$ be the SLP statement defining string t. We call the algorithm recursively to obtain the nonzeros of matrices $P^{\text{◲}}_{p,t'}$, $P^{\text{◰}}_{p,t'}$, $P^{\text{◳}}_{p,t'}$, $P^{\text{◲}}_{p,t''}$, $P^{\text{◰}}_{p,t''}$, $P^{\text{◳}}_{p,t''}$. The total number of nonzeros in each matrix triple is between m and $2m$. Conceptually, these matrices are submatrices of $P_{p,t'}$ over $\langle -m : n' \mid 0 : m + n' \rangle$, and $P_{p,t''}$ over $\langle -m : n'' \mid 0 : m + n'' \rangle$. However, the actual remapped index range after the recursive calls is $\langle -m : 2m \mid 0 : 3m \rangle$ for both matrix triples. We now compute the composition seaweed matrices $P^{\text{◲}}_{p,t}$, $P^{\text{◰}}_{p,t}$, $P^{\text{◳}}_{p,t}$ by Theorem 2. The total number of nonzeros in this matrix triple is again between m and $2m$. Conceptually, these matrices are submatrices of $P_{p,t}$ over $\langle -m : n \mid 0 : m + n \rangle$. However, the actual remapped index range after the composition is $\langle -m : 4m \mid 0 : 5m \rangle$. Therefore, there are at least $2m$ indices $\hat{\imath} \in \langle 0 : 4m \rangle$, such that the row $P^{\text{◲}}(\hat{\imath}, *)$ and the column $P^{\text{◰}}(*, \hat{\imath})$ both contain only zeros. We now delete exactly $2m$ such rows and columns from the respective matrices, and remap the index range to $\langle -m : 2m \mid 0 : 3m \rangle$, while preserving the linear order of the indices.

(End of recursive step.)

Second phase. We now have the nonzeros of the output matrices, remapped to the index range $\langle -m : 2m \mid 0 : 3m \rangle$. This is already sufficient to query the global LCS score, or substring-string LCS scores for p against t. However, if explicit indices of the nonzeros in the output seaweed matrices are required, the index range can be remapped back to $\langle -m : n \mid 0 : m + n \rangle$ by reversing every remapping step in the recursion.

Cost analysis.

First phase. The cost of a recursive step is dominated by the seaweed matrix composition, which runs in time $O(m \log m)$ by Corollary 2. There are $\bar{n}$ recursive steps in total, therefore the first phase runs in time $O(m \log m \cdot \bar{n})$.

Second phase. For each nonzero, the inverse remapping can be performed recursively in time $O(\bar{n})$. There are m nonzeros in total, therefore the second phase runs in time $O(m\bar{n})$.

Total. The overall running time is $O(m \log m \cdot \bar{n})$.

Algorithm 1 provides, as a special case, an algorithm for the LCS problem between a GC-string and a plain string, running in time $O(m \log m \cdot \bar{n})$; the LCS score can easily be queried from any one of the algorithm's three output matrices by Theorem 1.

Extensions. Hermelin et al. [6] considered the weighted alignment problem on a pair of GC-strings a, b of total compressed length $\bar{r} = \bar{m} + \bar{n}$, parameterised by the strings' total plain length $r = m + n$. In the case of rational weights, they gave an algorithm running in time $O(r^{1.2}\bar{r}^{1.4})$.

Based on the results of this section, this running time can be improved by the following straightforward algorithm. First, we uncompress one of the input strings — say, string b. Then, we run Algorithm 1 on the GC-string a as a text against the plain string b as a pattern. The resulting running time is $O(m \log m \cdot \bar{n}) = O(r \log r \cdot \bar{r})$. In an unpublished work by Hermelin et al. [5], this running time is further improved to $O\big(r \log(r/\bar{r}) \cdot \bar{r}\big)$.

3.2 Local Subsequence Recognition

The local subsequence recognition problem was introduced in Subsection 2.2 as a special case of the semi-local LCS problem. In the context of local subsequence recognition, a substring of text t is called a *matching substring*, if it contains the pattern p as a subsequence. A matching substring will be called *minimally matching*, if it is inclusion-minimal, i.e. it has no proper matching substring.

Local subsequence recognition can take the following forms: the *minimum-window subsequence recognition problem*, which asks for the locations of all substrings of t that are minimally matching, and the *fixed-window subsequence recognition problem*, which asks for the locations of all the matching substrings of a fixed length w. A combination of these two problems is the *bounded minimal-window subsequence recognition problem*, which asks for the locations of all the minimally matching substrings below a fixed length w.

Clearly, the output size for the described *reporting versions* of these problems may be exponential in $\bar{n}$; therefore, we have to parameterise the running time by the output size, which we denote by *output*. An algorithm for the reporting version of any of the above problems can typically be converted to solve the corresponding *counting version* of the problem. Such a counting algorithm, instead of reporting all the matching substrings, only returns their overall number. The running time of the counting versions for all algorithms described in this subsection will correspond to the running time of the reporting algorithm with $\mathit{output} = O(1)$.

Previous work. The minimal-window, fixed-window and bounded minimal-window subsequence recognition problems for a GC-text against a plain pattern have been considered by Cégielski et al. [3]. For each problem, they gave an algorithm running in time $O(m^2 \log m \cdot \bar{n} + \mathit{output})$. In [17], we gave an improved algorithm for these problems, running in time $O(m^{1.5}\bar{n})$.

New result. We now give a more efficient local subsequence recognition algorithm, running in time $O(m \log m \cdot \bar{n} + \mathit{output})$. The algorithm is based on Algorithm 1, which we extend as follows. In addition to the seaweed matrices

$P^{\text{▛}}_{p,t}$, $P^{\text{▜}}_{p,t}$, $P^{\text{▙}}_{p,t}$, we now also make use of the seaweed cross-matrix $P^{\text{■}}_{p,t',t''}$. We extend every recursive step by the reporting of minimally matching substrings that are cross-substrings in the current seaweed matrix composition.

Algorithm 2 (Local subsequence recognition).

Input: SLP of length $\bar{n}$, generating text t of length n; plain pattern p of length m.

Output: locations (or count) of minimally matching substrings in t.

Description. Similarly to Algorithm 1, index remapping has to be performed in the background in order to avoid an exponential growth of the indices. To simplify the exposition, we now assume constant-time index arithmetic, keeping the index remapping implicit.

First phase. Recursion on the input SLP generating t.

Recursion base: $n = \bar{n} = 1$. As in Algorithm 1, the seaweed matrices $P^{\text{▛}}_{p,t}$, $P^{\text{▜}}_{p,t}$, $P^{\text{▙}}_{p,t}$ can be computed by a linear sweep of string p. String t is matching, if and only if $m = 1$ and $t = p$; in this case, t is also minimally matching.

Recursive step: $n \geq \bar{n} > 1$. Let $t = t't''$ be the SLP statement defining string t. We run a recursive step of Algorithm 1, obtaining the seaweed matrices $P^{\text{▛}}_{p,t}$, $P^{\text{▜}}_{p,t}$, $P^{\text{▙}}_{p,t}$. In addition, we obtain the seaweed cross-matrix $P^{\text{■}}_{p,t',t''}$ by Theorem 2. This matrix has exactly m nonzeros. Let

$$\mathcal{L} = \left\{ \left(\hat{\imath}_{\frac{1}{2}}, \hat{\jmath}_{\frac{1}{2}}\right) \ll \left(\hat{\imath}_{\frac{3}{2}}, \hat{\jmath}_{\frac{3}{2}}\right) \ll \cdots \ll \left(\hat{\imath}_{s-\frac{1}{2}}, \hat{\jmath}_{s-\frac{1}{2}}\right) \right\}$$

be the $\ll$-chain of all $\lessgtr$-maximal nonzeros in $P^{\text{■}}_{p,t',t''}$, where $s = |\mathcal{L}| \leq m$.

By Theorem 1, a substring $t\langle i : j \rangle$ is matching, if and only if $P^{T\Sigma T}_{p,t}(i,j) = 0$, i.e. the point (i,j) is not $\lessgtr$-dominated by any nonzeros in the seaweed matrix $P_{p,t}$. Recall that a substring $t\langle i : j \rangle$ is a *cross-substring*, if $i \in [0 : n' - 1]$, $j \in [n'+1 : n]$; in other words, a cross-substring consists of a non-empty suffix of t' and a non-empty prefix of t''. A point (i,j) corresponding to a cross-substring can only be $\lessgtr$-dominated by nonzeros within the seaweed cross-matrix $P^{\text{■}}_{p,t',t''}$. Therefore, a cross-substring $t\langle i : j \rangle$ is matching, if and only if point (i,j) is not $\lessgtr$-dominated by any of the nonzeros in $P^{\text{■}}_{p,t',t''}$, or, equivalently, by any point in $\mathcal{L}$.

Consider the set of all points in $[-m : n' \mid n' : m + n]$, not $\lessgtr$-dominated by any point in $\mathcal{L}$. The $\lessgtr$-minimal points in this set, excluding the irrelevant boundary points $\left(\lfloor \hat{\imath}_{\frac{1}{2}} \rfloor, n'\right)$ and $\left(n', \lceil \hat{\jmath}_{s-\frac{1}{2}} \rceil\right)$, are interleaved with the points of $\mathcal{L}$, and form themselves a $\ll$-chain of size $s - 1$:

$$\mathcal{M} = \left\{ \left(\lfloor \hat{\imath}_{\frac{3}{2}} \rfloor, \lceil \hat{\jmath}_{\frac{1}{2}} \rceil\right) \ll \left(\lfloor \hat{\imath}_{\frac{5}{2}} \rfloor, \lceil \hat{\jmath}_{\frac{3}{2}} \rceil\right) \ll \cdots \ll \left(\lfloor \hat{\imath}_{s-\frac{1}{2}} \rfloor, \lceil \hat{\jmath}_{s-\frac{3}{2}} \rceil\right) \right\}$$

Let $i \in [0 : n' - 1]$, $j \in [n' + 1 : n]$. Then, a cross-substring $t\langle i : j \rangle$ is minimally matching, if and only if $(i,j) \in \mathcal{M}$. The number of such points (i,j) is at most

$|\mathcal{M}| = m - 1$ (it could be strictly less, since some points in $\mathcal{M}$ may lie outside the range $[0 : n' - 1 \mid n' + 1 : n]$, and therefore not correspond to any cross-substrings).

(End of recursive step)

Second phase. For every SLP symbol, we now have the locations of its minimally matching cross-substrings. Furthermore, every non-trivial substring of t corresponds to a cross-substring for some SLP symbol, under an appropriate transformation of indices. By another recursion on the structure of the SLP, it is now straightforward to obtain either the locations or the count of all the minimally matching substrings in t.

Cost analysis.

First phase. As in Algorithm 1, each seaweed matrix composition runs in time $O(m \log m)$. The $\lll$-chains $\mathcal{L}$ and $\mathcal{M}$ can be obtained in time $O(m)$. Hence, the running time of a recursive step is $O(m \log m)$. There are $\bar{n}$ recursive steps in total, therefore the whole recursion runs in time $O(m \log m \cdot \bar{n})$.

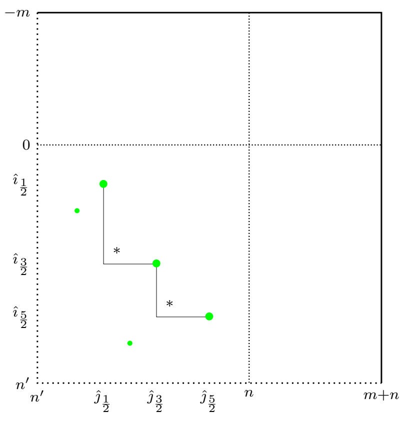
Second phase. For every SLP symbol, there are at most $m-1$ minimally matching cross-substrings. Given the output of the first phase, the locations of all minimally matching substrings in t can be reported in time $O(m\bar{n} + \mathit{output})$.

Total. The overall running time is $O(m \log m \cdot \bar{n} + \mathit{output})$.

Example 6. Figure 3 shows a snapshot of a recursive step in the first phase of Algorithm 2. Subfigure 3a shows the seaweed cross-matrix $P^{\blacksquare}_{p,t',t''}$; in this particular example, all its nonzeros belong to the string-substring seaweed matrix $P^{\boxdot}_{p,t',t''}$. Subfigure 3b shows the corresponding seaweed braid. Matrix $P^{\blacksquare}_{p,t',t''}$ contains $m = 5$ nonzeros, shown by green bullets in Subfigure 3a, and by green seaweeds in Subfigure 3b. Out of these five nonzeros, three are $\lesseqgtr$-maximal; they are shown by larger bullets (respectively, by thicker seaweeds). The three $\lesseqgtr$-maximal nonzeros form the $\lll$-chain $\mathcal{L}$. Consequently, there are $3 - 1 = 2$ points in the interleaved $\lll$-chain $\mathcal{M}$. In Subfigure 3a, these two points are shown by asterisks; in Subfigure 3b, the corresponding two substrings of t are shown by dotted brackets. The interleaving between $\lll$-chains $\mathcal{L}$ and $\mathcal{M}$ is shown in Subfigure 3a by solid black lines. Both points of $\mathcal{M}$ lie within the range $[0 : n' - 1 \mid n' + 1 : n]$, and therefore each of them corresponds to a minimally matching cross-substring in t.

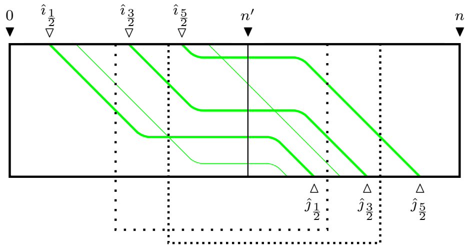
By Theorem 1, a substring in t is matching, if and only if the corresponding rectangle in the alignment dag is not pierced by a seaweed entering at its left-hand boundary and leaving at its right-hand boundary. Notice that the bracketed substrings of t in Figure 3b are exactly the two inclusion-minimal cross-substrings satisfying this property.

Algorithms for the fixed-window and the bounded minimal-window subsequence recognition problems can be obtained by straightforward modifications of Algorithm 2; see [14] for details. The running time of both modifications is still $O(m \log m \cdot \bar{n} + \mathit{output})$.



$-m$
0
$\hat{\imath}_{\frac{1}{2}}$
$\hat{\imath}_{\frac{3}{2}}$
$\hat{\imath}_{\frac{5}{2}}$
n'
n'
$\hat{\jmath}_{\frac{1}{2}}$
$\hat{\jmath}_{\frac{3}{2}}$
$\hat{\jmath}_{\frac{5}{2}}$
n
$m+n$
*
*

(a) Seaweed cross-matrix $P^{\blacksquare}_{p,t',t''}$ and $\ll$-chain of $\lessgtr$-maximal nonzeros



0
$\hat{\imath}_{\frac{1}{2}}$
$\hat{\imath}_{\frac{3}{2}}$
$\hat{\imath}_{\frac{5}{2}}$
n'
n
$\hat{\jmath}_{\frac{1}{2}}$
$\hat{\jmath}_{\frac{3}{2}}$
$\hat{\jmath}_{\frac{5}{2}}$

(b) Corresponding seaweed braid

Fig. 3. A snapshot of Algorithm 2 (local subsequence recognition)

4 Conclusions

Using the techniques of semi-local string comparison and fast seaweed matrix composition, we have obtained improved algorithms for LCS computation and local subsequence recognition between a GC-text and a plain pattern. The local subsequence recognition problem can be regarded as a rudimentary form of the *approximate matching problem*, which asks for substrings in the text that are close to the pattern in terms of the *edit distance*. In a wider context presented in [14], we extend the techniques of the current paper to solve this more general problem.

It remains an open problem whether our results can be extended to other compression models, e.g. LZ77 [21] or collage systems [7].

References

1. Amir, A., Benson, G., Farach, M.: Let sleeping files lie: Pattern matching in Z-compressed files. Journal of Computer and System Sciences 52(2), 299–307 (1996)
2. Bille, P., Farach-Colton, M.: Fast and compact regular expression matching. Theoretical Computer Science 409(3), 486–496 (2008)

3. Cégielski, P., Guessarian, I., Lifshits, Y., Matiyasevich, Y.V.: Window subsequence problems for compressed texts. In: Grigoriev, D., Harrison, J., Hirsch, E.A. (eds.) CSR 2006. LNCS, vol. 3967, pp. 127–136. Springer, Heidelberg (2006)
4. Crochemore, M., Landau, G.M., Ziv-Ukelson, M.: A subquadratic sequence alignment algorithm for unrestricted score matrices. SIAM Journal on Computing 32(6), 1654–1673 (2003)
5. Hermelin, D., Landau, G.M., Landau, S., Weimann, O.: Unified compression-based acceleration of edit-distance computation. Technical Report 1004.1194, arXiv
6. Hermelin, D., Landau, G.M., Landau, S., Weimann, O.: A unified algorithm for accelerating edit-distance computation via text-compression. In: Proceedings of the 26th STACS, pp. 529–540 (2009)
7. Kida, T., Matsumoto, T., Shibata, Y., Takeda, M., Shinohara, A., Arikawa, S.: Collage system: A unifying framework for compressed pattern matching. Theoretical Computer Science 298(1), 253–272 (2003)
8. Lifshits, Y.: Processing compressed texts: A tractability border. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 228–240. Springer, Heidelberg (2007)
9. Lifshits, Y., Lohrey, M.: Querying and embedding compressed texts. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 681–692. Springer, Heidelberg (2006)
10. Masek, W.J., Paterson, M.S.: A faster algorithm computing string edit distances. Journal of Computer and System Sciences 20, 18–31 (1980)
11. Myers, G.: Approximately matching context-free languages. Information Processing Letters 54, 85–92 (1995)
12. Needleman, S.B., Wunsch, C.D.: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology 48(3), 443–453 (1970)
13. Rytter, W.: Algorithms on compressed strings and arrays. In: Bartosek, M., Tel, G., Pavelka, J. (eds.) SOFSEM 1999. LNCS, vol. 1725, pp. 48–65. Springer, Heidelberg (1999)
14. Tiskin, A.: Semi-local string comparison: Algorithmic techniques and applications. Technical Report 0707.3619, arXiv
15. Tiskin, A.: Semi-local longest common subsequences in subquadratic time. Journal of Discrete Algorithms 6(4), 570–581 (2008)
16. Tiskin, A.: Semi-local string comparison: Algorithmic techniques and applications. Mathematics in Computer Science 1(4), 571–603 (2008)
17. Tiskin, A.: Faster subsequence recognition in compressed strings. Journal of Mathematical Sciences 158(5), 759–769 (2009)
18. Tiskin, A.: Fast distance multiplication of unit-Monge matrices. In: Proceedings of ACM–SIAM SODA, pp. 1287–1296 (2010)
19. Wagner, R.A., Fischer, M.J.: The string-to-string correction problem. Journal of the ACM 21(1), 168–173 (1974)
20. Welch, T.A.: A technique for high-performance data compression. Computer 17(6), 8–19 (1984)
21. Ziv, G., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory 23, 337–343 (1977)
22. Ziv, G., Lempel, A.: Compression of individual sequences via variable-rate coding. IEEE Transactions on Information Theory 24, 530–536 (1978)

The Optimal Strategy for the Average Long-Lived Consensus[⋆]

Eric Rémila

Université de Lyon,
Laboratoire de l'Informatique du Parallélisme,
(umr 5668 CNRS - ENS de Lyon - Université Lyon 1),
site Monod, ENS de Lyon,
46 allée d'Italie, 69364 Lyon Cedex 7 - France
`eric.remila@ens-lyon.fr`

Abstract. Consider a system composed of n sensors operating in synchronous rounds. In each round an *input vector* of sensor readings x is produced, where the r-th entry of x is a value, selected in a finite set of potential values, produced by the r-th sensor. The sequence of input vectors is assumed to be *smooth*: exactly one entry of the vector changes from one round to the next one. The system implements a fault-tolerant averaging *consensus function* f. This function returns, in each round, a representative *output value* v of the sensor readings x. Assuming there are $a + 1$ equal entries of the vector, f is required to return a value that appears at least $a + 1$ times in x.

We study strategies that minimize the *instability* of a fault-tolerant consensus system. More precisely, we find the strategy that minimizes, in average, the frequency of output changes over a random walk sequence on input vectors (where each component of the vector corresponds to a particular sensor reading).

1 Introduction

Consider a system composed of n sensors sampled at synchronous rounds. In each round an *input vector* of sensor readings is produced, where the r-th entry of the vector is a value from some finite set V produced by the r-th sensor. To simplify the presentation, the sampling interval is assumed to be short enough, to guarantee that the sequence of input vectors is *smooth*: exactly one entry of a vector changes from one round to the next one.

There are situations where, for fault-tolerant purposes, a number of sensors are placed in the same location. Ideally, in such cases, all sensor readings should be equal. But this is not always the case; discrepancies may arise due to differences in sensor readings or to malfunction of some sensors. Thus, the system must implement some form of fault-tolerant averaging *consensus function* f that

⋆ This work is partially supported by Programs Ecos C09E04 and IXXI (Complex System Institute, Lyon).

returns a representative *output value* of the sensor readings. Assuming that there are $a+1$ equal entries of the vector, f is required to return a value that appears in more than a entries of x.

The same questions arise when consensus is done not between the values of sensors, but between the opinions of actors. Suppose for example that you have a server which can give several types of data to a bunch of clients. At a given time, each client has a favorite type of data it wants to receive, but the server can only broadcast one type of data to all the clients. If there is a cost to switching between requests (say, because one can no longer use cached data), then in order to serve as much clients as possible in the long-run, it might be wise to sometimes give a content that fewer of them want, but which we have already started serving.

In a social setting, the same kind of question arises whenever a group has to make a consensual decision. For example, consider a disc-jockey in a wedding party. There are both older people, who fancy dancing to a nice waltz, and younger ones, eager to get their kicks on techno music. Our disc-jockey has to make sure that the dance-floor is never too empty according to who is ready to dance at a given time. But if he changes the music too often, then nobody is going to be happy: stability matters. More seriously, in an election system, one might want to have a decision that is at the same time representative and stable, so that the policies which are decided have the time to be applied (for a caricatural example, the decision between war or peace needs some stability). In a setting where there is no term-mandate and decision-making is done live, we show that the stability can be enforced through election rules (i.e., the decision function).

In this context, the most natural function f is the one that returns the most common value of vector x. However, the *instability* of such function is high. In fact, as the next example shows ($n = 5$ and $a = 1$), the output value computed by this f could change from one round to the next one unnecessarily often:

$$\begin{array}{ll} \text{inputs:} & 00011 \rightarrow 10011 \rightarrow 10010 \rightarrow 11010 \rightarrow \cdots \\ \text{outputs:} & \quad\;\; 0 \rightarrow \qquad 1 \rightarrow \qquad 0 \rightarrow \qquad 1 \rightarrow \cdots \end{array}$$

If instead of the previous f we consider the one that decides the smallest value in x that appears at least $a+1$ times, then no output changes would have occurred in the previous sequence (in the example $0 < 1$ and $a + 1 = 2$). Moreover, in order to reduce further the instability, we could consider a function that tries to stay with the output of previous rounds.

The worst case instability of consensus functions was studied in two previous papers [4,7]. The input sequence considered in those papers was assumed to be, in addition to smooth, *geodesic*: the r-th entry of the input vector was allowed to change *at most once* over the sequence. The instability of a consensus function was given by the largest number of output changes over any such sequence, called a *geodesic path*. Notice that a geodesic path must be finite, since the set V from which the input vectors draw their values is finite. The case $V = \{0, 1\}$ of binary input vectors was considered in [7]. The case of multi-valued input vectors, where the set V is arbitrary, turned out to be much more difficult and required higher-dimensional topological methods [4]. Recently, instability under byzantine failures was studied in [6].

The notion of geodesic stability is not totally satisfying. For example, for $a = 0$, the function consisting in following the result of a fixed sensor has a minimal instability. In [1] , we introduced, as an alternative measure, a more natural (and subtle) notion called *average instability.* We removed the geodesic requirement and therefore the smooth sequences of input vectors we considered were random walks over the hypercube. If $P = x_0, x_1, \ldots$ is such a walk, then the average instability of a consensus function f is given by the fraction of time f changes its output over P.

We studied in [1] the case when the input is binary ($S = \{0, 1\}$), and found optimal strategies, without memory and with memory. In [13], we analyzed the three value case ($S = \{0, 1, 2\}$), when the process has no memory. In the present paper, we treat the general case ($S = \{0, 1, 2, ...m - 1\}$) with finite memory. We prove that a very simple function, only taking account the current input vector and the previous consensus value is actually optimal. This proves that a small memory (of size $\log m$) is sufficient, in order to maximize the stability.

The paper is divided as follows. In section 2, we give formal definitions of concepts above, and present our optimal solution, and our result. In section 3, we introduce a lightly different consensus model, the auxiliary process, which enforces symmetry properties. In this framework, we prove that our solution is optimal, even if a finite horizon is fixed (i.e. random walks of fixed finite length are used). In section 4, we use a technique using the law of large numbers to deduce our result from the result about the auxiliary process.

As noted in [7], studying the instability of consensus functions may have applications in various areas of distributed computing, such as self-stabilization [5] (indeed, see [9]), Byzantine agreement [2], real-time systems [10], complexity theory [8] (boolean functions), and VLSI energy saving [3,11,14] (minimizing the number of transitions).

2 Average Instability

2.1 Framework

Let n, m be positive integers and a be a non negative integer such that $n > m\,a$. The set $= \{0, 1, ..., m-1\}$ is denoted by $\mathbb{Z}_m$. The *input space* of dimension n, is a graph whose vertex set is $V_n = (\mathbb{Z}_m)^n$, called *input vectors*. The edges E_n are all (unordered) pairs of vertices whose vectors differ in exactly one component. We assume that an initial probability distribution, denoted by λ is fixed on V_n.

The *distance* $d(x_1, x_2)$ between two input vectors x_1, x_2 is equal to the number of entries in which they differ. Thus, $d(x_1, x_2) = d$ if and only if the shortest path between x_1 and x_2 in V_n is of length d.

For any input vector x and $b \in \mathbb{Z}_m$, we denote by $\#_b(x)$ the number of entries in x that are equal to b. We also denote by $dom(x)$ the lowest integer b such that $\#_b(x)$ is maximal.

A *system* is a tuple $D = (m, n, a, S, \tau, f)$, where S is a finite set called the *memory* that includes a special initial symbol $\perp \in S$, $\tau : V_n \times S \to S$ is the memory function, and $f : V_n \times S \to \mathbb{Z}_m$ is the decision function.

Threshold condition: The fault-tolerance requirement that f must satisfy is the following. With the notation above, we must have:

$$f(x, s) = b \Rightarrow \#_b(x) > a.$$

We say that f is a *consensus function* when it satisfies the threshold constraint above.

A pair (x, s) is a *configuration* of the system. Each configuration is given a consensus value $b = f(x, s)$ An *execution* of the system is a sequence $(x_0, s_0) \to (x_1, s_1) \to \ldots$, where $s_0 = \perp$, x_{k+1} is a neighbor of x_k, and $s_{k+1} = \tau(x_k, s_k)$.

We assume that if x is the current input vector, then the next input vector x' is taken in a random uniform way from the vectors at distance one from x in the hypercube. The initial input vector is chosen according to some distribution λ. Once the initial state $s_0 = \perp$ is determined, so is the initial configuration, (x_0, s_0). The next configuration is produced by choosing at random a neighbor of x_0, say x_1, and we get the next configuration (x_1, s_1), where $s_1 = \tau(x_0, s_0)$.

Formally, we have a Markov process (P, λ) whose set of states is V_n and there is a transition from x to x' if $\{x, x'\} \in E_n$. The probability of such a transition is $\frac{1}{(m-1)n}$, which defines the transition matrix P. The initial vector distribution is λ.

Let X_k denote the random variable of this process after k steps. We successively define the random variables $S_0, S_1, ...$ by $S_0 = \perp$ and $S_{k+1} = \tau(X_k, S_k)$. We also state $Z_k = (X_k, S_k)$, $B_k = f(Z_k) = f(X_k, S_k)$ and

$$F_p(D) = \frac{1}{p} \sum_{k=0}^{p-1} \delta(B_k, B_{k+1})$$

(where δ is the function defined on $(\mathbb{Z}_m)^2$ by $\delta(s, s') = 1$ when $s = s'$, and $\delta(s, s') = 0$ otherwise). Thus, $F_p(D)$ gives the change frequency during a walk of length p.

The sequence $Z_0, Z_1, ...$ is given by the Markov process (P', μ_0) whose set of states is $V_n \times S$ and there is a transition from (x, s) to (x', s') if $\{x, x'\} \in E_n$ and $\tau(x, s) = s'$ with probability $\frac{1}{(m-1)n}$. The probability μ_0 is the probability product $\lambda \bigotimes \delta_\perp$ where $\delta_\perp$ denotes the Dirac distribution on S concentrated in $\perp$, (i.e. the unique probability on S such that $\delta_\perp(\perp) = 1$).

We give here a version of the ergodic theorem on Markov chains, which will be used below. We recall that a Markov chain is *irreducible* if it is possible to get to any state from any state by a sequence of transitions, and a probability distribution is *invariant* (or *stationary*) if the distribution does not change when a step of the Markov chain is executed. It is well known that an irreducible Markov chains admits a unique invariant distribution.

Theorem 1. *Consider an irreducible discrete time Markov process $(Z_n)_{n\in N}$ on a finite space V. For any bounded function $f : V \to \mathbb{R}$, we have*

$$\mathbb{P}\Big(\lim_{l\to\infty}(\frac{1}{p}\sum_{k=0}^{p-1} f(Z_k)) = \sum_{v\in V}\pi_v f(v)\Big) = 1$$

where $(\pi_v)_{v\in V}$ denotes the unique invariant distribution.

Therefore, even without the irreducibility hypothesis, there exists a value $\overline{f}$ such that we have

$$\mathbb{P}\Big(\lim_{p\to\infty}(\frac{1}{p}\sum_{k=0}^{p-1} f(Z_k)) = \overline{f}\Big) = 1.$$

The first part is a classical version of the ergodic theorem, the second part is trivially obtained by decomposition of the probability distribution of X_0 on irreducible components. The main idea is that the computation average on random walk is reduced to the computation of the stationary distribution.

Proposition 1. *For any consensus system D, The* average instability *of D, defined by:*

$$\mathrm{inst}(D) = \mathbb{E}(\lim_{p\to\infty} F_p(D))$$

exists and does not depend on the initial distribution λ.

Moreover we have,

$$\mathrm{inst}(D) = \lim_{p\to\infty} \mathbb{E}(F_p(D))$$

Proof. Consider the Markov chain in which each state is an ordered couple $((x,s),(x',s'))$ of $(V_n \times S)^2$ such that $\{x,x'\}$ is and edge of E_n, $s' = \tau(x,s)$, and there is a transition of probability $\frac{1}{(m-1)n}$ from each state $((x,s),(x',s'))$ to each state $((y,u),(y',u'))$ such that $(x',s') = (y,u)$. The function defined over the set of states (the arcs) is $\phi((x,s),(x',s')) = \delta(s,s')$. Applying Theorem 1 (with $\mu_0 \bigotimes \mathbb{P}(\mu_0)$ as origin distribution), we get the first part of the proposition.

The second part is a direct application of the famous Lebesgue's dominated convergence theorem.

In particular, for the study of the instability, the theorem above allows to assume without loss of generality that the distribution λ is the uniform distribution on V_n. We will work with this hypothesis in the remaining of the paper.

2.2 The Result

Let f_0 denotes function, defined on $V_n \times (\mathbb{Z}_m \cup \{\perp\})$, by

- $f_0(x,\perp) = \mathrm{dom}(x)$.
- if $b \in \mathbb{Z}_m$ and $\#_b(x) > a$, then $f_0(x,b) = b$,
- if $b \in \mathbb{Z}_m$ and $\#_b(x) \leq a$, then $f_0(x,b) = \mathrm{dom}(x)$,

Informally, f_0 encodes the laziest natural strategy: the consensus value remains unchanged as long as its is possible. When it has to be flipped, the dominant value is taken. Notice that the only information stored is an element of $\mathbb{Z}_m$, whose size does not depend on the number n of sensors. This result below proves that a surplus of memory is useless.

Theorem 2. *Let D_0 be the consensus system: $D_0 = (m, n, a, \mathbb{Z}_m \cup \{\perp\}, f_0, f_0)$ For any system $D = (k, n, t, S, \tau, f)$ satisfying the threshold condition, we have*

$$\text{inst}(D) \geq \text{inst}(D_0).$$

Clearly, the ideal best strategy *with a fortune-teller* would be to keep previous consensus as long as possible and, being forced to change it, to ask the fortune-teller which of the new acceptable values will have the longest lifetime. The theorem shows that we can get rid of such a fortune-teller.

If the result is quite natural, the proof is far from trivial. In a first step, we need to introduce an auxiliary modified process, which enforces symmetries. Then we study the (equivalent of) instability with a finite fixed number of steps, for the auxiliary process (section 3). Symmetries added allow us to prove that the function given above is the optimal one for instability with a finite horizon. This part contains most of original ideas of the paper.

Afterwards (section 4), we use a technique using the law of large numbers to deduce the theorem, as it is stated above, from the previous results on the auxiliary process.

3 The Auxiliary Process and Its Optimal Strategy

3.1 The Auxiliary Process

Let $D = (m, n, a, S, \tau, f)$ be a system. The corresponding auxiliary process is given by a uniform random walk on the (undirected) graph (V'_n, E'_n) with $V'_n = V_n \cup E_n$, and each element $\{e, x\}$ of E'_n is formed from an input vector x of V_n and an edge e of E_n containing x. Informally an additional node is placed in each edge. When the walk reaches such a node, an endpoint of the edge is chosen, uniformly at random, for the following position of the walk, while the memory remains unchanged.

Let $e = \{x, x'\}$ be an edge of E_n. Assume that $x = (b_1, b_2, ...b_n)$ and $x' = (b'_1, b'_2, ...b'_n)$. the edge e can be seen as an n-uple $e = (c_1, c_2, ..., c_n)$ with, for $1 \leq r \leq n$, $c_r = b_r$ when $b_r = b'_r$ and c_r is the unordered pair $\{b_r, b'_r\}$ when $b_r \neq b'_r$. Thus, exactly one component of the n-uple e is a pair. For each b of $\mathbb{Z}_m$, we define $\#_b(e)$ by $\#_b(e) = (\#_b(x) + \#_b(x'))/2$.

Let $X'_0, E_0, X'_1, E_1, X'_2, E_2, ...$ be the random variables giving the successive positions during the uniform random walk on (V'_n, E'_n) (with the distribution of X'_0 being uniform on V_n). We define the random variable S'_k by:

- if $X'_{k+1} = X'_k$, then $S'_{k+1} = S_k$,
- if $X'_{k+1} \neq X'_k$, then $S'_{k+1} = \tau(S_k)$.

We also define $B'_k = f(X'_k, S'_k)$ and the random variable

$$F'_p(D) = \frac{1}{p}\sum_{k=0}^{p-1} \delta(B'_k, B'_{k+1}),$$

in a similar way than in the original process.

3.2 Trajectory Colorings

For the study of the modified process, it is easier to allow consensus strategies which depend on the whole history. This is why we present the following framework.

A *trajectory* of length p is a $2p+1$-uple $t = (x_0, e_0, x_1, e_1, ..., x_{p-1}, e_{p-1}, x_p)$ such that, for each integer k such that $0 \leq k < p$, $\{x_k, e_k\}$ and $\{e_k, x_{k+1}\}$ both are elements of E'_n (it may happen that $x_k = x_{k+1}$). A trajectory has the *median property* (which does not exist in the original process): let k and k' be integers such that $0 \leq k < k' \leq p$, and b and b' be elements of S such that $\#_b(x_k) > \#_{b'}(x_k)$ and $\#_b(x_{k'}) < \#_{b'}(x_{k'})$. There exists an integer k'' such that $k \leq k'' \leq k'$ and either $\#_b(x_{k''}) = \#_{b'}(x_{k''})$ or $\#_b(e_{k''}) = \#_{b'}(e_{k''})$. The median property is a main reason for introducing the auxiliary process.

For $k \leq p$, the prefix of length k of t, denoted by $\text{pref}_k(t)$, is the trajectory $\text{pref}_k(t) = (x_0, e_0, x_1, e_1, ..., x_k)$. The set of trajectories of length p is denoted by T_p, and the set of trajectories of length at most p is denoted by $T_{\leq p}$. Hence $T_{\leq p} = \bigcup_{0 \leq k \leq p} T_k = \bigcup_{0 \leq k \leq p, T \in T_p} \text{pref}_k(t)$. The set of all trajectories is denoted by T.

Given a consensus system $D = (m, t, a, S, f, \tau)$, the *memory function* on trajectories is the function τ': $T \to S$, inductively defined by:

- $\tau'(x_0) = \tau(x_0, \perp)$
- $\tau'(x_0, e_0, x_1, e_1, ..., x_{k+1}) = \tau(x_{k+1}, \tau'(x_0, e_0, x_1, e_1, ..., x_k))$ if $x_{k+1} \neq x_k$,
- $\tau'(x_0, e_0, x_1, e_1, ..., x_{k+1}) = \tau'(x_0, e_0, x_1, e_1, ..., x_k)$ otherwise.

For the auxiliary process, it is easier to work with consensus values possibly depending on the whole trajectory, and not only on a finite memory. In order do it, we extend the notion of consensus function by introducing colorings.

A *trajectory coloring* g is a function $T_p \to \mathbb{Z}_m$. Informally, the notion of coloring allows to extend consensus values without paying attention in states. A coloring g is called a *consensus coloring* if it satisfies the following threshold condition: for each trajectory $(x_0, e_0, x_1, e_1, ..., x_k)$, we have: $g(x_0, e_0, x_1, e_1, ..., x_k) = b \Rightarrow \#_b(x_k) > a$.

The trajectory coloring given by D is the function g_D inductively defined by:

- $g_D(x_0, e_0) = f(x_0, \perp)$
- $g_D(x_0, e_0, x_1, e_1, ..., x_{k+1}) = f(x_{k+1}, \tau'(x_0, e_0, x_1, e_1, ..., x_k, e_k))$.

Obviously, g_D is a consensus coloring, since f is consensus function. For short, we state: $g_0 = g_{D_0}$.

For each trajectory $t = (x_0, e_0, x_1, e_1, ..., x_p)$, each trajectory coloring g, and each integer k such that $0 \leq k < p$, we state:

$$\delta_g(k, t) = \delta(g(\text{pref}_k(t)), g(\text{pref}_{k+1}(t))) \quad \text{and} \quad \text{change}_g(t) = \sum_{k=0}^{p-1} \delta_g(k, t).$$

When $\delta_g(k, t) = 1$, we say that we have a *g-flip* in position k.

For $p > 0$, we define the random variable:

$$F''_p(g) = \text{change}_g(X'_0, E'_0, ..., X'_p)/p.$$

We have: $F''_p(g_D) = F'_p(D)$. If $(x_0, e_0, x_1, e_1, ..., x_p)$ and $(x'_0, e'_0, x'_1, e'_1, ..., x'_p)$ are trajectories, then $\mathbb{P}((X'_0, E'_0, ..., X'_p) = (x_0, e_0, ..., x_p)) = \mathbb{P}((X'_0, E'_0, ..., X'_p) = (x'_0, e'_0, ..., x'_p))$, which induces:

$$\mathbb{E}(F''_p(g)) = \frac{1}{p|T_p|} \sum_{t \in T_p} \text{change}_g(t)$$

3.3 Analysis of the Auxiliary Process

Let g and g' be two different colorings; the distance $d(g, g')$ between g and g' is $2^{-i(g,g')}$ where $i(g, g')$ denotes the length of the shortest trajectory t such that $g(t) \neq g'(t)$ (we also state $d(g, g) = 0$).

Lemma 1. *For any consensus coloring g such that $d(g, g_0) \leq 2^{-p}$, there exists a consensus coloring g' such that $\mathbb{E}(F''_p(g')) \leq \mathbb{E}(F''_p(g))$ and $d(g_0, g') < d(g_0, g)$.*

Proof. Let i such that $d(g, g_0) = 2^{-i}$, with $i \leq p$. In particular, for any trajectory t of length at least i, we have: $g(\text{pref}_{i-1}(t)) = g_0(\text{pref}_{i-1}(t))$.

We have to exhibit a consensus coloring g' such that, for each trajectory t of $T_{\leq i}$, $g'(t) = g_0(t)$, and $\sum_{t \in T_p} \text{change}_{g'}(t) \leq \sum_{t \in T_p} \text{change}_g(t)$. For any trajectory t of length at most i, we are forced to state $g'(t) = g_0(t)$.

Now take a trajectory $t = (x_0, e_0, x_1, e_1, ..., x_j,)$ with $j > i$. we have to define $g'(t)$. In order to make main ideas clearly appear, we assume in first step that $i \geq 1$. The case when $i = 0$ will be treated at the end.

The first idea for the construction of g' is to change nothing when the situation is OK. Formally, if $g(\text{pref}_i(t)) = g_0(\text{pref}_i(t))$, then we state $g(t) = g'(t)$. Thus, when $j = p$, we have: $\text{change}_g(t) = \text{change}_{g'}(t)$.

The second idea for the construction of g' is to postpone the g-flip in position $i - 1$, when it is possible. Formally, if $\#_{g(\text{pref}_{i-1}(t))}(x_i) > a$, (i.e. if it is possible to have $\delta_{g'}(i-1, t) = 0$), then we state $g'(t) = g(\text{pref}_{i-1}(t))$ if t is of length $j = i$ (in other words, we enforce that $\delta_{g'}(i - 1, t) = 0$) and $g(t) = g'(t)$ if $j > i$.

When $\delta_g(i - 1, t) = 0$, using by the first idea above, we already know that $\text{change}_g(t) = \text{change}_{g'}(t)$. Otherwise, $\delta_g(i - 1, t) = 1$ while $\delta_{g'}(i - 1, t) = 0$, and the index is i is the only other one for which it may happen that $\delta_g(k, t) \neq \delta_{g'}(k, t)$. Thus, when $j = p$, we have: $\text{change}_g(t) \leq \text{change}_{g'}(t)$.

The difficult case arises when we have a trajectory t with the property that: $\#_{g(\mathrm{pref}_{i-1}(t))}(x_i) = a$, which enforces that we will necessarily have $\delta_{g'}(i-1, t) = 1$, and $g(\mathrm{pref}_i(t)) \neq \mathrm{dom}(x_i)$ (thus $g(\mathrm{pref}_i(t)) \neq g_0(\mathrm{pref}_i(t))$.

To avoid abuse of notations, we will assume without loss of generality that $g(\mathrm{pref}_{i-1}(t)) = 0$, $g(\mathrm{pref}_i(t)) = 1$ and $\mathrm{dom}(x_i) = 2$.

– Assume in a first step that, $i \leq k \leq j$. We have $\#_2(x_k) > \#_1(x_k)$. In this case, we state:
  - $g'(t) = 2$ if $g(t) = 1$,
  - $g'(t) = g(t)$ otherwise.

  Informally, the idea is that the value 1 can be replaced by 2, after x_i.
  When $g(t) = 1$, we have: $\#_2(x_j) > \#_1(x_j) > a$. When $g(t) \neq 1$, we have: $g'(t) = g(t) > a$. Thus g' satisfies the property of consensus coloring for t.
  For each trajectory t of T_p of this case, the equality: $\mathrm{change}_{g'}(\mathrm{pref}_i(t)) = \mathrm{change}_g(\mathrm{pref}_i(t))$ holds, and, for any position k such that $k \geq i$, we have $\delta_{g'}(k, t) \leq \delta_g(k, t)$. Thus: $\mathrm{change}_{g'}(t) \leq \mathrm{change}_g(t)$.

– When the hypothesis above is not satisfied, the direct comparison of $\mathrm{change}_{g'}$ (t) with $\mathrm{change}_g(t)$ fails. So we use a method which matches trajectories for which roles of 1 and 2 are inverted. It is the most subtle idea of the paper. The value $\mathrm{change}_g(t)$ is compared with $\mathrm{change}_g(\mu(t))$, where $\mu(t)$ is trajectory deduced from t, by a 'mirror technique".
  From the median property, there exists an integer q such that $q > i$ and either $\#_1(e_q) = \#_2(e_q)$ or $\#_1(x_q) = \#_2(x_q)$. We take the smallest q satisfying this condition. Informally, q is the first index after i for which the value 1 and 2 play the same role for consensus.
  We treat the case when $\#_1(x_q) = \#_2(x_q)$, (the case when $\#_1(e_q) = \#_2(e_q)$ being similar). Formally, let us state: $x_q = (b_{q,1}, b_{q,2}, \ldots. b_{q,n})$, and let B_1 (respectively B_2) be the set of indices r such that $b_{q,r} = 1$ (respectively $b_{q,r} = 2$). We have $card(B_1) = card(B_2)$, so we can fix a bijection h from B_1 to B_2, only depending on x_q. Let σ be the permutation of $\{1, 2, \ldots.., n\}$, such that:
  - $\sigma(r) = h(r)$ when $r \in B_1$,
  - $\sigma(r) = h^{-1}(r)$ when $r \in B_2$,
  - $\sigma(r) = r$ otherwise.

  Let $\sigma_{1,2}$ be the permutation of $\mathbb{Z}_m$, such that $\sigma_{1,2}(1) = 2$, $\sigma_{1,2}(2) = 1$, and $\sigma_{1,2}(b) = b$ otherwise. We have $\sigma^2 = id_n$ (where id_n is the identical permutation on $\{1, 2, \ldots.., n\}$) and $(\sigma_{1,2})^2 = id_{\mathbb{Z}_m}$, the identical permutation on $\mathbb{Z}_m$.
  For the trajectory $t = (x_0, e_0, x_1, e_1, \ldots., x_j)$, we define the trajectory $\mu(t) = (x'_0, e'_0, , \ldots.., x'_j)$ as follows:
  - $\mathrm{pref}_q(\mu(t)) = \mathrm{pref}_q(t)$,
  - for $q \leq k \leq j$, if $x_k = (b_{k,1}, b_{k,2}, \ldots, b_{k,n})$ and $x'_k = (b'_{k,1}, b'_{k,2}, \ldots, b'_{k,n})$, then have: $b'_{k,r} = \sigma_{1,2}(b_{k,\sigma(r)})$.
  - for $q \leq k < j$, if $e_k = (c_{k,1}, c_{k,2}, \ldots, c_{k,n})$ and $e'_k = (c'_{k,1}, c'_{k,2}, \ldots, c'_{k,n})$, then have:

  * $c'_{k,r} = \sigma_{1,2}(c_{k,\sigma(r)})$ when $c_{k,\sigma(r)}$ is not a pair,
  * $c'_{k,r} = \{\sigma_{1,2}(b), \sigma_{1,2}(b')\}$ in the case when $c_{k,\sigma(r)}$ is the pair $\{b, b'\}$.

Informally, the idea is that the sensor r has a referent, which is $\sigma(r)$. For the construction of $\mu(t)$, r simulates what is done by its referent, except for values 1 and 2 which are transposed.

One easily checks that $\mu(t)$ is really a trajectory: if x_k and e_k only differ on the index r for which $c_{k,r} = \{b_{k,r}, b'\}$, then x'_k and e'_k only differ on the index $\sigma(r)$ for which $c'_{k,\sigma(r)} = \{\sigma_{1,2}(b_{k,\sigma(r)}), \sigma_{1,2}(b')\} = \{b'_{k,\sigma(r)}, \sigma_{1,2}(b')\}$. A similar argument holds for checking the transition from e'_k to x'_{k+1}.
For each trajectory t of this case, we state:

- $g'(t) = \sigma_{1,2}(g(\mu(t)))$.

We have $\#_{g(\mu(t))}(x_j) > a$, thus $\#_{\sigma_{1,2}(g(\mu(t)))}(x'_j) > a$, i.e. $\#_{g'(t)}(x'_j) > a$, which ensures that g' satisfies the threshold condition for t.
We have $\text{change}_{g'}(\text{pref}_i(t)) = \text{change}_g(\text{pref}_i(t)) = \text{change}_g(\text{pref}_i(\mu(t))$. For $i \le k < q$, $\text{pref}_k(t)$ satisfies the previous item ($q = +\infty$), thus $\delta_{g'}(k,t) \le \delta_g(k,t) = \delta_g(k, \mu(t))$ For $q \le k < j$, we have, by symmetry : $\delta_{g'}(k,t) = \delta_g(k, \mu(t))$. Thus $\text{change}_{g'}(t) \le \text{change}_g(\mu(t))$.
On the other hand, we have: $\mu(\mu(t)) = t$, which guarantees that μ is a bijective mapping on the set T' of trajectories t starting by $(x_0, e_0, x_1, e_1, ..., x_k)$. It follows that

$$\sum_{t \in T' \cap T_p} \text{change}_{g'}(t) \le \sum_{t \in T' \cap T_p} \text{change}_g(\mu(t))$$

and

$$\sum_{t \in T' \cap T_p} \text{change}_g(\mu(t)) = \sum_{t' \in \mu(T' \cap T_p)} \text{change}_g(t') = \sum_{t \in T' \cap T_p} \text{change}_g(t)$$

which gives:

$$\sum_{t \in T' \cap T_p} \text{change}_{g'}(t) \le \sum_{t \in T' \cap T_p} \text{change}_g(t)$$

Adding on all possible sets T', the analysis above gives: $\sum_{t \in T_p} \text{change}_{g'}(t) \le \sum_{t \in T_p} \text{change}_g(t)$, i.e. the result.

For $i = 0$, the analysis is completely similar to the case when i is positive with the properties: $\#_{g(\text{pref}_{i-1}(t))}(x_i) = a$, and $g(\text{pref}_i(t)) \neq \text{dom}(x_i)$.

As a corollary, we get the optimality result below.

Proposition 2. *For any consensus system $D = (m, n, a, S, f, \tau)$ and any positive integer p, we have $\mathbb{E}(F'_p(D)) \ge \mathbb{E}(F'_p(D_0))$.*

4 From the Auxiliary Process to the Original Process

For the average point of view, Proposition 2 insures that D_0 is definitely the best consensus system, for the auxiliary process. We will now deduce that D_0 is also the best one for the original process, at least in an asymptotic way.

We first present the ideas in a informal way. Take any consensus system D, consider, for the auxiliary process, trajectories of length $2p$ and study the number of input changes done during these trajectories (we have an input change when $x'_k \neq x'_{k+1}$). In average, half of time the trajectory remains in the same state, and half of the time an input change occurs. Therefore, for p large, the number of input changes is close to p, with high probability. Thus, for p large, the number of output changes in trajectories of the auxiliary process of length $2p$ has approximately the same distribution than the number of output changes in the p first steps of the original process. Thus $\mathbb{E}(F'_{2p}(D))$ and $\mathbb{E}(F_p(D))/2$ are nearly equal, and have the same limit when p tends to infinite. By this way, asymptotic comparisons of systems are in the same sense in the auxiliary and in the original process.

The intuitive ideas above can be formalized by the following proposition.

Proposition 3. *For any consensus system D, $\lim_{p\to\infty} \mathbb{E}(F'_{2p}(D))$ exists and*

$$\lim_{p\to\infty} \mathbb{E}(F'_{2p}(D)) = \frac{\mathrm{inst}(D)}{2}$$

Proof. We fix a consensus system D. We lighten notations, stating, for each $p > 0$, $G_p = pF_p(D)$, $G'_p = pF'_p(D)$, (i.e. G_p and G'_p are the number of state changes, respectively in a walk of length p in the original process, and in trajectory of length p in the auxiliary process).

We also introduce N_p, the number of input changes during a trajectory of length p in the auxiliary process. Formally: $N_0 = 0$, $N_{k+1} = N_k$ if $X'_{k+1} = X'_k$, and $N_{k+1} = N_k + 1$ if $X'_{k+1} \neq X'_k$. The distribution of N_p is the binomial law $B(p, \frac{1}{2})$, i.e. for $0 \leq k \leq p$, we have $\mathbb{P}(N_p = k) = \frac{\binom{p}{k}}{2^p}$.

Remark the conditional random variable $G'_p|(N_p = k)$ has the same distribution than the random variable G_k. Thus, since $F'_{2p}(D) = \frac{G'_{2p}}{2p}$, we have:

$$\mathbb{E}(F'_{2p}(D)) = \frac{1}{2p}\sum_{k=0}^{2p} \mathbb{P}(N_{2p} = k)\mathbb{E}(G'_{2p}|(N_{2p} = k)) = \frac{1}{2p}\sum_{k=0}^{2p} \mathbb{P}(N_{2p} = k)\mathbb{E}(G_k).$$

Thus

$$\mathbb{E}(F'_{2p}(D)) = \sum_{k=0}^{2p} \mathbb{P}(N_{2p} = k)\mathbb{E}(\frac{G_k}{k})\frac{k}{2p} = \sum_{k=0}^{2p} \frac{k}{2p}\mathbb{P}(N_{2p} = k)\mathbb{E}(F_k(D)).$$

The relation above gives a first link between average of variables F'_k and variables F_k.

Let ϵ denote a positive real (assumed to be small). We split the sum above into three parts. The main idea is the fact that, for p large, $\frac{N_p}{p}$ is approximatively equal to $\frac{1}{2}$ with high probability. We first have:

$$\sum_{k=0}^{\lfloor(1-\epsilon)p\rfloor} \frac{k}{2p}\mathbb{P}(N_{2p}=k)\mathbb{E}(F_k(D)) \leq \sum_{k=0}^{\lfloor(1-\epsilon)p\rfloor} \mathbb{P}(N_{2p}=k) \leq \mathbb{P}(N_{2p} \leq (1-\epsilon)p)$$

Secondly, we have:

$$\sum_{k=\lceil(1+\epsilon)p\rceil}^{2p} \frac{k}{2p}\mathbb{P}(N_{2p}=k)\mathbb{E}(F_k(D)) \leq \sum_{k=\lceil(1+\epsilon)p\rceil}^{2p} \mathbb{P}(N_{2p}=k) \leq \mathbb{P}(N_{2p} \geq (1+\epsilon)p)$$

The law of large numbers tells that, for any pair (δ, ϵ) of positive integers, we have, for p sufficiently large, $\mathbb{P}(\frac{N_{2p}}{2p} < \frac{1}{2} - \delta) \leq \epsilon$. Thus, taking $\delta = \frac{\epsilon}{2}$, we get: $\mathbb{P}(N_{2p} \leq (1-\epsilon)p) \leq \epsilon$. On the other hand, by symmetry, we have: $\mathbb{P}(N_{2p} \leq (1-\epsilon)p) = \mathbb{P}(N_{2p} \geq (1+\epsilon)p)$. Therefore, we obtain:

$$0 \leq \sum_{k=0}^{\lfloor(1-\epsilon)p\rfloor} \frac{k}{2p}\mathbb{P}(N_{2p}=k)\mathbb{E}(F_k(D)) + \sum_{k=\lceil(1+\epsilon)p\rceil}^{2p} \frac{k}{2p}\mathbb{P}(N_{2p}=k)\mathbb{E}(F_k(D)) \leq 2\epsilon$$

The central term is a bit more difficult to study. For p sufficiently large, we have: $k \geq (1-\epsilon)p \Rightarrow |\mathbb{E}(F_k(D)) - \text{inst}(D)| \leq \epsilon$, from Proposition 1. Thus we get

$$\sum_{k=\lfloor(1-\epsilon)p\rfloor+1}^{\lceil(1+\epsilon)p\rceil-1} (\frac{1-\epsilon}{2})\,\mathbb{P}(N_{2p}=k)\,(\text{inst}(D)-\epsilon) \leq \sum_{k=\lfloor(1-\epsilon)p\rfloor+1}^{\lceil(1+\epsilon)p\rceil-1} \frac{k}{2p}\,\mathbb{P}(N_{2p}=k)\,\mathbb{E}(F_k(D))$$

i. e.

$$(\frac{1-\epsilon}{2})\,(\text{inst}(D)-\epsilon)\,\mathbb{P}((1-\epsilon)p < N_{2p} < (1+\epsilon)p) \leq \sum_{k=\lfloor(1-\epsilon)p\rfloor+1}^{\lceil(1+\epsilon)p\rceil-1} \frac{k}{2p}\,\mathbb{P}(N_{2p}=k)\,\mathbb{E}(F_k(D))$$

Finally, using the law of large number, we get, for p sufficiently large:

$$(\frac{1-\epsilon}{2})\,(\text{inst}(D)-\epsilon)\,(1-2\epsilon) \leq \sum_{k=\lfloor(1-\epsilon)p\rfloor+1}^{\lceil(1+\epsilon)p\rceil-1} \frac{k}{2p}\,\mathbb{P}(N_{2p}=k)\,\mathbb{E}(F_k(D))$$

By similar arguments, we have:

$$\sum_{k=\lfloor(1-\epsilon)p\rfloor+1}^{\lceil(1+\epsilon)p\rceil-1} \frac{k}{2p}\,\mathbb{P}(N_{2p}=k)\,\mathbb{E}(F_k(D)) \leq (\frac{1+\epsilon}{2})\,(\text{inst}(D)+\epsilon)$$

Thus, reconstructing the whole sum by adding the three parts, we get:

$$(\frac{1-\epsilon}{2})(\text{inst}(D)-\epsilon)(1-2\epsilon)\leq\sum_{k=0}^{2p}\frac{k}{2p}\mathbb{P}(N_{2p}=k)\mathbb{E}(F_k(D))\leq(\frac{1+\epsilon}{2})(\text{inst}(D)+\epsilon)+2\epsilon$$

i.e.

$$(\frac{1-\epsilon}{2})(\text{inst}(D)-\epsilon)(1-2\epsilon)\leq\mathbb{E}(F'_{2p}(D))\leq(\frac{1+\epsilon}{2})(\text{inst}(D)+\epsilon)+2\epsilon$$

which ensures the result of the proposition.

We now have the material to easily prove Theorem 2: for any consensus system D and any $p>0$, we have $\mathbb{E}(F'_p(D))\geq\mathbb{E}(F'_p(D_0))$ from Proposition 2. Thus, $\lim_{p\to\infty}\mathbb{E}(F'_p(D))\geq\lim_{p\to\infty}\mathbb{E}(F'_p(D_0))$, which gives $\text{inst}(D)\geq\text{inst}(D_0)$, from Proposition 3.

5 Extensions

5.1 Infinite Memory

If we allow the memory S to be infinite, $\text{inst}(D)$ might not exist, but the average $\mathbb{E}(\liminf_{p\to\infty}F_p(D))$ always exists, and can be considered as a lower bound for the cost. The theorem below proves that there is no hope to find a better consensus function, even with an infinite memory.

Theorem 3. *For any system $D=(k,n,t,S,\tau,f)$ satisfying the threshold condition (with S being eventually infinite), we have:*

$$\mathbb{E}(\liminf_{p\to\infty}F_p(D))\geq\liminf_{p\to\infty}\mathbb{E}(F_p(D))\geq\text{inst}(D_0).$$

Proof. (sketch) The first inequality is classical in integration theory. Proposition 2 holds even when S is infinite, and Proposition 3 can be adapted for inferior limits, when S is infinite, which gives the result.

5.2 Other Probability Distributions

The analysis above can be criticized for its particularity. The first particularity is the fact of the initial distribution is uniform. But general theorems on Markov chains prove that the instability does not depends on the initial distribution. In other words, any initial distribution can be chosen to get the same results.

In a lot of cases, the use of the uniform random Markov chain seems to be unadapted For instance, consider a scenario where the inputs are heat sensors, indicating either "hot" or "cold". If most of them indicate that the ambient temperature is indeed "hot", we would expect that it is the truth and there is a higher transitional probability for changing from "cold" to "hot" than the other way around (and vice versa). At the opposite, in politics, if a large majority for one side does exist, then it appears that people do not hesitate before leaving this majority and we would expect a higher transitional probability for changing from the majority to the opposition than in the other way.

In our analysis, the key-point is the introduction of the coupling of trajectories with mirror-trajectories. This technique can be applied as soon as the trajectory and its mirror have the same probability. This is guaranteed when there is a symmetry of inputs (which can be seen as the anonymousness of agents).

Aknowledgements. The author is very grateful with Florent Becker and Ivan Rapaport for helpful discussions in Santiago de Chile.

References

1. Becker, F., Rajsbaum, S., Rapaport, I., Rémila, E.: Average Binary Long-Lived Consensus: Quantifying the Stabilizing Role Played by Memory. Theoretical Computer Science 411(6), 1558–1566 (2010)
2. Berman, P., Garay, J.: Cloture votes: $n/4$-resilient distributed consensus in $t+1$ rounds. Math. Sys. Theory 26(1), 3–19 (1993)
3. Chandrakasan, A.P., Brodersen, R.W.: Low power digital CMOS design. Kluwer Academic Publishers, Dordrecht (1995)
4. Davidovitch, L., Dolev, S., Rajsbaum, S.: Stability of Multi-Valued Continuous Consensus. SIAM J. on Computing 37(4), 1057–1076 (2007)
5. Dolev, S.: Self-Stabilization. MIT Press, Cambridge (2000)
6. Dolev, D., Hoch, E.N.: OCD: obsessive consensus disorder (or repetitive consensus). In: Proc. of the 27th Annual ACM Symp. on Principles of Distributed Computing, PODC 2008, pp. 395–404 (2008)
7. Dolev, S., Rajsbaum, S.: Stability of Long-lived Consensus. J. of Computer and System Sciences 67(1), 26–45 (2000); Preliminary version in Proc. of the 19th Annual ACM Symp. on Principles of Distributed Computing, (PODC 2000), pp. 309–318 (2000)
8. Kahn, J., Kalai, G., Linial, N.: The Influence of Variables on Boolean Functions. In: Proc. of the IEEE FOCS, pp. 68–80 (1988)
9. Kutten, S., Masuzawa, T.: Output Stability Versus Time Till Output. In: Pelc, A. (ed.) DISC 2007. LNCS, vol. 4731, pp. 343–357. Springer, Heidelberg (2007)
10. Kopetz, H., Veríssimo, P.: Real Time and Dependability Concepts. In: Mullender, S. (ed.) Distributed Systems, ch. 16, pp. 411–446. ACM Press, New York (1993)
11. Musoll, E., Lang, T., Cortadella, J.: Exploiting the locality of memory references to reduce the address bus energy. In: Proc. of the Int. Symp. on Low Power Electronics and Design, pp. 202–207 (August 1997)
12. Norris, J.R.: Markov Chains. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge (1998)
13. Rapaport, I., Rémila, E.: Average memoryless long-lived Consensus: the three value case. In: Patt-Shamir, B., Ekim, T. (eds.) SIROCCO 2010. LNCS, vol. 6058, pp. 114–126. Springer, Heidelberg (2010)
14. Su, C.-L., Tsui, C.-Y., Despain, A.M.: Saving power in the control path of embedded processors. IEEE Design & Test of Comp., 24–30 (1994)

Improved Online Scheduling in Maximizing Throughput of Equal Length Jobs

Thang Nguyen Kim

LAMSADE, Université Paris Dauphine, France

Abstract. Motivated by issues raised from data broadcast and networks using ATM and TCP/IP, we consider an online scheduling problem on a single machine. In the problem, each job i is revealed at release time r_i, has processing time p_i, deadline d_i and weight w_i. Preemption is allowed and there are two models of preemption: preemption with restart and preemption with resume. The goal is to maximize the *throughput* — the total weight of all jobs completed on time. In the paper, we consider the problem where all processing time of jobs are equal and present improved algorithms which achieve 4.24-competitive in both models of preemption.

1 Introduction

Data broadcast involves information distribution from a server to clients. The advantage of broadcasting technologies is that different users having the same request can be simultaneously satisfied by a broadcast. Information lies in a large range, from movies, soccer matches to stock market news, etc. Clients are also diverse and have different interests in certain moments. Hence, to maximize a given *quality of service* it is allowed that the server interrupts the currently broadcast page and starts a new one. Nevertheless, to satisfy a previously-interrupted page, the server has to broadcast again from the beginning.

ATM network has been designed to send telephone, radio, television communication as well as usual network data. In networks using ATM and TCP/IP, IP packets have to be split into small ATM cells and fed into the ATM networks. In general, packet sizes are bounded by the capacity of Ethernet, i.e. 1500 bytes, and in many cases they have the same length which equals the maximal capacity. The network transmits cells separately. In maximizing a given *quality of service*, it is allowed to stop transmitting cells of some packets and start sending cells of other ones. However, in contrast to data broadcast, in order to complete packets that still have remaining cells unsent, the network only needs to transmit the remaining instead of starting from the beginning.

Problem definition. These applications can be formulated as an online scheduling problem on a single machine where each job i arrives online at its release time r_i, has processing time p_i, deadline d_i and weight w_i. The job's parameters are unknown until its arrival. All these quantities except possibly w_i are integer. The objective is to maximize *throughput*, which is the total weight of jobs completed on time. Preemption of jobs is allowed. Motivated by the above applications, we consider two models of preemption in the problem:

(i) the *preemptive model with restart*, where a job can be interrupted, but when it is scheduled again, it must be scheduled from the beginning
(ii) the *preemptive model with resume*, where in contrast, when a job is scheduled again, the previously done work can be resumed.

The problem under these two models of preemption can be denoted as $1|\text{online}-r_i|\sum w_i(1-U_i)$ and $1|\text{online}-r_i; pmtn|\sum w_i(1-U_i)$, respectively according to notation in [3].

1.1 Related Work

It is known that, in both models of preemption, the deterministic competitive ratio is unbounded if jobs' processing times are arbitrary [1]. Dürr et al. [6] presented an algorithm that gave a tight bound $\Theta(p/\log p)$ in both models where the processing time of all jobs are bounded by p.

In the model of preemption with restart, a 5-competitive algorithm is given in [9,2]. Zheng et al. [12] provided an improved algorithm Bar that was 4.56-competitive for jobs with equal processing time, arbitrary weights. Chrobak et al. [5] gave a tight 1.5-competitive deterministic algorithm for equal processing time, unit weight jobs.

In the model of preemption with resume, the best known upper bound and lower bound competitive ratio for the case of equal length and arbitrary weight jobs are 5 [6] and 2.59 [2,6], respectively. For an interesting special case where jobs have unit processing time (i.e. $p_i = 1 \ \forall i$), the problem (related to buffer management for packet switches [8]) is widely studied, and the deterministic competitive ratio lies between $\phi(\approx 1.618)$ [4] and 1.83 [7]. Another direction of research is to consider resource augmentation, and in [10] a deterministic online algorithm was presented which has constant competitive ratio provided that the algorithm is allowed a constant speedup of its machine compared to the adversary.

1.2 Our Contribution

In the paper, we study the problem with jobs of equal processing time, arbitrary weight. The variant of equal processing time jobs has been widely studied (see [11, chapter 14]). We give algorithms which are 4.24-competitive for both models of preemption. The algorithms are essentially the same and the analysis are based on charging schemes. Roughly speaking, the issue is to solve the dilemma of choice between a lower-weighted job with imminent deadline and a higher-weighted job with later deadline.

In Section 2, we recall some standard notions. Then we illustrate the ideas that inspire the improved algorithms. In Section 3, we consider the model of preemption with restart in which we present the algorithm together with its intuition and the analysis. Even though the improvement is small, the algorithm helps in designing an algorithm with the same competitive ratio in the model of preemption with resume. In Section 4, we describe a 4.24-competitive algorithm in the latter model. The structure of the proof is similar to the one in the former model. However, the charging scheme is more subtle since the adversary may schedule jobs in different pieces and it presents the main difficulty of the proof in the model of preemption with resume.

2 Preliminaries

An algorithm is *r-competitive* if for any job sequence released by an adversary, the algorithm gain is at least r-fraction of the optimal offline solution where the whole sequence of jobs is known in advance.

Let p be the jobs' processing time, i.e. $p_i = p \ \forall i$. Consider an algorithm. Let C_i be the completion time of job i by the algorithm. Let $q_i(t)$ be the remaining processing time of job i for the algorithm at time t. When there is no confusion, we simply write q_i. In preemption with restart, if a job i is interrupted at some time t then $q_i(t+1) = p$.

We say that an algorithm schedules a job at time t meaning that the algorithm executes unit of this job in interval $[t, t+1)$. A job i is *pending* for the algorithm at time t if it has not been completed before and $r_i \leq t$ and $t + q_i(t) \leq d_i$. A job i is *urgent* at time t if $d_i < t + q_i(t) + p$.

In the model of preemption with restart, let $S_i(t)$ be the latest moment before t that an algorithm starts job i. In the model of preemption with resume, $S_i(t)$ denotes the latest moment $\tau < t$ such that at time τ, the algorithm schedules job i but at the previous moment $(\tau - 1)$, the algorithm schedules other job or it is an idle-time. Again, when there is no confusion, we simply write S_i.

Without loss of generality, assume that the adversary starts a job if and only if it will complete the job; and the adversary schedules jobs in Earliest Deadline First manner. In the analysis, we abbreviate the algorithm and the adversary by Alg and Adv, respectively.

Starting point of improved algorithms. Consider the following algorithm for the model of preemption with restart. If there is no currently scheduled job, schedule the pending one with highest weight. Otherwise, if there is a new job i arriving with weight at least twice that of the currently scheduled job then interrupt the latter and schedule i.

This algorithm is 5-competitive [9,2]. Observe that the algorithm considers the jobs' deadlines only in verifying whether jobs are pending; it totally ignores the correlation of that important parameter among the jobs. Hence, a better algorithm should be more involved in the deadlines of jobs, not only in verifying the pending property. An idea of improvement is that if a new released job is urgent, even if its weight is not large, one may delay the execution of the currently scheduled job and schedule the new job. However, postponing the heavy currently scheduled job might result in a lost of throughput if some new heavier job will be released later. A treatment is presented in [12] in which they handled implicitly the jobs' deadlines by turning them into a function of weight. In the paper, we deal explicitly with jobs' deadlines and present new algorithms with improved bounds.

3 A 4.24-Competitive Algorithm for Preemption with Restart

The algorithm A. Let $1 < \beta < 3/2$ be a constant to be defined later. Initially, set $Q := \emptyset$ and $\alpha := 0$. Throughout the execution of the algorithm, at any time, Q stores the last job interrupted according to condition $[A2]$ below.

At time t, if there is either no currently scheduled job or a job completion, then schedule the heaviest pending job. Otherwise, let j be the currently scheduled job. If there is no new released job, then continue to schedule j. Otherwise, let i be a new released job with heaviest weight. Job j is interrupted if one of the following conditions holds.

$A1$. **if:** $w_i \geq 2w_j$ and $w_i \geq 2^{\alpha} w(Q)$ where $w(Q)$ is the weight of job in Q. (Note that if $Q = \emptyset$ then $w(Q) = 0$ so the second inequality requirement is always satisfied.)
do: Schedule job i. Set $\alpha := 0$ and $Q = \emptyset$.

$A2$. **if:** $\alpha = 0$, $\beta w_j \leq w_i \leq 2w_j$, i is urgent and j can be scheduled later, i.e. $d_j \geq t + 2p$.
do: Schedule job i^* which is the heaviest among all urgent jobs, i.e., $i^* = \arg\max\{w_\ell : d_\ell < t + 2p\}$. Set $\alpha = 1$ and $Q := \{j\}$.

$A3$. **if:** i is urgent, $w_i \geq 2w_j + w_{j'}$ where j' is the job previously interrupted by j and there is no job ℓ satisfying the following conditions:

$$S_j(t) + 2p \leq d_\ell < t + 2p \quad \text{and} \quad w_\ell \geq w_j.$$

do: Schedule job i.

At any interruption, if $\alpha \geq 1$ then $\alpha := \alpha + 1$. If a job is completed then set $\alpha = 0$ and $Q = \emptyset$. Conventionally, if an interruption satisfies both conditions $[A1]$ and $[A3]$ then we refer that interruption to $[A1]$.

Informally, the counter α indicates whether there exists an interruption of type $[A2]$ which is not followed by an $A1$-interruption or a job completion. If there exists, α also indicates the number of job interruptions from the last job q interrupted according to $[A2]$ and the set Q stores job q.

Intuition of the algorithm.

– The first condition $[A1]$ lies in the same spirit as the 5-competitive algorithms: if there is a new released job with heavy enough weight then schedule that new job. In the condition, we need one more inequality ($w_i \geq 2^{\alpha} w_Q$) compared to the 5-competitive algorithm to ensure that w_i can compensate previous interruptions of different types.

- The second condition [$A2$] represents the idea that it is possible to delay the currently scheduled job if a new urgent job arrives even with non-heavy weight. However, we do not want to interrupt many consecutive jobs due to this condition since that may result in a small throughput. It is the reason we introduce a control variable α and condition [$A2$] depends on α. Precisely, the algorithm does not allow two $A2$-interruptions without an $A1$-interruption or a job completion in between.
- The purpose of the last condition [$A3$] is to handle a situation in which a new urgent heavy job arrives after an $A2$-interruption, but not heavy enough to interrupt the currently scheduled job according to [$A1$]. If there exists a job ℓ with properties described in [$A3$] then scheduling i is not profitable since we can schedule ℓ later and scheduling i means that ℓ will be dropped out. Otherwise, it is beneficial to interrupt j and schedule i.

First, we make some observations before presenting the analysis of the algorithm.

Observation 1. *1. Before any $A2$-interruption is either a job completion or an $A1$-interruption.*

2. Consider a sequence of consecutive $A3$-interruptions. Then, before the first ($A3$-) interruption of the sequence is an $A2$-interruption. Moreover, there are at most two $A3$-interruptions in the sequence. Consequently, α is always at most 3.

Proof. 1. By contradiction, suppose that i_1 is interrupted by i_2 and i_2 is interrupted according to condition [$A2$] by i_3. Then the interruption of i_1 by i_2 is not of type [$A2$] since otherwise counter $\alpha \geq 1$ so i_3 cannot interrupt i_2 according to condition [$A2$]. Moreover, if i_2 interrupts i_1 according to [$A3$], meaning i_2 is urgent then i_3 cannot interrupt i_2 according to condition [$A2$] because i_2 may not be scheduled later. Therefore, the interruption between i_1 and i_2 is of type [$A1$].

2. Before the first interruption (of type [$A3$]) in the sequence must be an interruption of type [$A2$] since otherwise that $A3$-interruption would have been referred to an interruption of type [$A1$] as convention. We argue the second claim by contradiction. Suppose that there are n jobs $i_n, i_{n-1}, \ldots, i_0$ where $n \geq 3$ and i_ℓ interrupt $i_{\ell+1}$ according to [$A3$] for $0 \leq \ell \leq n-1$. Hence, by previous claim, there exits a job i_{n+1} which is interrupted by i_n according to [$A2$]. We argue that in fact, i_0 is heavy enough to interrupt i_1 according to [$A1$]. As $n \geq 3$, we have:

$$\begin{aligned} w_{i_{n-3}} &\geq 2w_{i_{n-2}} + w_{i_{n-1}} \geq 2(2w_{i_{n-1}} + w_{i_n}) + w_{i_{n-1}} \\ &\geq 5(2w_{i_n} + w_{i_{n+1}}) + 2w_{i_n} > 2^4 w_{i_n} \end{aligned}$$

where the inequalities are due to condition of [$A3$] and $w_{i_n} > w_{i_{n+1}}$. Hence, i_{n-3} satisfies condition [$A1$] so the interruption between i_{n-3} and i_{n-2} would have been referred to [$A1$] by convention (contradiction).

The claim that $\alpha \leq 3$ is straightforward from previous observations. □

Lemma 1. *Consider a sequence of jobs $m, \dots, 1, 0$ where job ℓ interrupts job $(\ell+1)$ for $1 \leq \ell \leq m-1$ and job 0 interrupts job 1 according to condition $[A1]$. Then*

1. *$w_\ell \leq w_0 \cdot 2^{-\ell}$ for $0 \leq \ell \leq m$.*
2. *$w_{\ell'} \leq w_0 \cdot 2^{-\ell}$ where ℓ' be a job started by* Adv *in $[S_{\ell+1}, S_\ell)$ and has not been completed by* Alg*. Consequently, the total weight of jobs which have not been completed by* Alg *and are started by* Adv *in $[S_m, S_0)$ is bounded by $2w_0 - 2w_m$.*

Proof. 1. We prove the claim by induction. It is straightforward for $\ell = 0$. Suppose that $w_\ell \leq w_0 \cdot 2^{-\ell}$. If job ℓ interrupts job $(\ell+1)$ according to $[A1]$ or $[A3]$ then $w_{\ell+1} \leq w_\ell \cdot 2^{-1} \leq w_0 \cdot 2^{-\ell+1}$. The remaining case is that ℓ interrupts $(\ell+1)$ according to $[A2]$. Consider the first $A1$-interruption after time S_ℓ. Assume that it is the interruption between jobs h and $(h+1)$. This interruption exists since $h = 0$ is a candidate. By definition of $[A1]$, $w_\ell \leq w_h \cdot 2^{h-\ell}$ (since $\alpha = \ell - h$). Moreover, by induction hypothesis, $w_h \leq w_0 \cdot 2^{-h}$. Therefore, $w_\ell \leq w_0 \cdot 2^{-\ell}$.

2. By contradiction, let ℓ be the largest index such that there exists a job ℓ' started by Adv in $[S_{\ell+1}, S_\ell)$, $w_{\ell'} > w_0 \cdot 2^{-\ell}$ and ℓ' is not completed before by Alg. Hence, for any job $h > \ell$, $w_h \leq w_0 \cdot 2^{-h} < w_{\ell'} \cdot 2^{\ell-h}$. As ℓ' has not been completed by Alg, ℓ' may interrupt job $(\ell+1)$ according to $[A1]$. Hence, job ℓ' is job ℓ. This contradicts that $w_\ell \leq w_0 \cdot 2^{-\ell} < w_{\ell'}$.
   The total weighs of jobs which have not been completed by Alg and are started by Adv in $[S_m, S_0)$ is bounded by $\sum_{\ell=0}^{m-1} w_0 \cdot 2^{-\ell} \leq 2w_0 - 2^{-m}w_0 \leq 2w_0 - 2w_m$. □

Analysis. We analyze the competitiveness of the algorithm by a charging scheme. For convenience we renumber the jobs completed by the algorithm from 1 to n, such that the completion times are ordered $C_1 < \dots < C_n$. Also we denote $C_0 = 0$. We divide the schedule of the algorithm into *phases* $[C_{i-1}, C_i)$, for $i = 1, \dots, n$. We say that $[C_{i-1}, C_i)$ is the *phase of job i* for $i = 1, \dots, n$. Consider the phase of job i. Let $f(i)$ be the first job started by Alg in this phase. Remark that, $f(i)$ is not necessarily completed by Alg.

The Charging Scheme.

1. If a job i is scheduled by Adv and i has been completed by Alg before then charge w_i to job i (self-charge).
2. If a job j is started by Adv in the phase of job i and j has not been completed before by Alg, then charge w_j to i.
3. For each phase of job i, if there exists a next phase of job $i+1$ such that $S_{f(i+1)} = C_i$, i.e. meaning no idle-time between jobs i and $f(i+1)$, then charge $2w_{f(i+1)}$ from job i to job $i+1$.

Informally, in the first two steps of the charging scheme, we charge the total throughput of Adv to jobs which are completed by Alg. In the third step, we

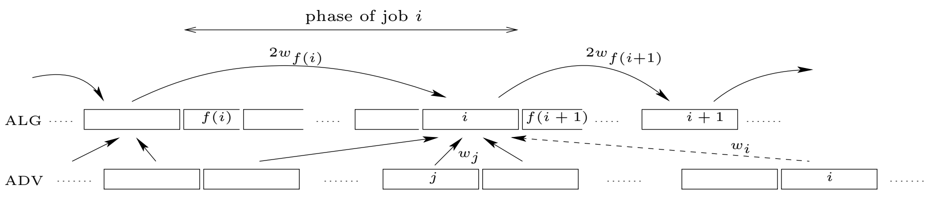


Fig. 1. Illustration of the charging scheme. The dashed pointer and the curly pointer represent the self-charge and the charge of step 3, respectively.

redistribute the charges among the latter so that: each of such jobs receives a charge within factor r of its weight where r is the desired competitive ratio.

Consider the phase of job i and let i' be the job started by the ALG just after finishing i. Note that in case there exists no such job i', conventionally $w_{i'} = 0$; in case i' exists, $i' = f(i+1)$. In the analysis, we will argue that the total charge that i receives before step 3 of the charging scheme is at most $r \cdot w_i - 2w_{f(i)} + 2w_{i'}$ where r is a constant revealed later. In the redistribution step (step 3), each job i transfers $2w_{i'}$ to other job and possibly receives $2w_{f(i)}$. So the total charge that each job i completed by ALG receives is at most $r \cdot w_i$, which deduces the competitive ratio r of the algorithm. Hence, it is sufficient to prove the bound of job i's charge before step 3 by the term above. To simplify the exposition, until the end of the section, we refer the charge of a job as the amount that the job receives before redistribution (before step 3).

We say that a phase is of *type 1, 2, 3 or 0* if the last interruption in the phase is according to $[A1]$, $[A2]$, $[A3]$ or there is no interruption in the phase, respectively. If the phase of job i is 0-type then i receives at most one charge from a job j started in $[S_i, C_i)$ and probably one self-charge. As j does not interrupt i and j is not completed before, $w_j < 2w_i$. Hence, the total charge that i receives is at most $3w_i$. In the following, we bound the charge when the job i's phase is of type $1, 2$ and 3.

Lemma 2. *If the phase of job i is of type 1 then the charge that i receives is at most $(3+\beta)w_i - 2w_{f(i)} + 2w_{i'}$.*

Proof. By Lemma 1, the charge that i receives from jobs started before S_i in ADV is at most $2w_i - 2w_{f(i)}$. Consider the job j scheduled by ADV in $[S_i, C_i)$. We have $w_j < 2w_i$ since otherwise, j can interrupt i by $[A1]$. If $w_j \leq \beta w_i$ or j has been completed by ALG or i receives no self-charge then the total charge that i receives is at most $\max\{2w_i - 2w_{f(i)} + w_i + \beta w_i, 2w_i - 2w_{f(i)} + w_i, 2w_i - 2w_{f(i)} + w_j\} \leq (3+\beta)w_i - 2w_{f(i)}$. The remaining case is that $\beta w_i < w_j < 2w_i$ and i receives a self-charge. Let τ be the moment that ADV starts j.

1. **j is not urgent at τ**
   Then j is still pending at completion time of i, so $w_{i'} \geq w_j$. Hence, the charge that i receives in this case is $2w_i - 2w_{f(i)} + w_i + w_j \leq 3w_i - 2w_{f(i)} + w_{i'}$.
2. **j is urgent at τ**

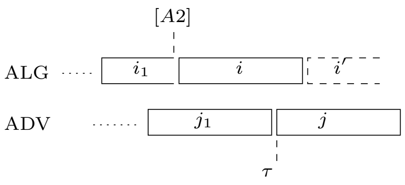


Fig. 2. Illustration of a phase of type 2

Since i receives a self-charge, $d_i \geq \tau + 2p$. As j is urgent and $w_j > \beta w_i$, i would have been interrupted by j according to condition $[A2]$, contradicts that the phase is of type 1. □

Lemma 3. *If the phase of job i is of type 2 then the charge that i receives is at most* $\max\left\{\frac{6}{\beta} - 1, 4\right\} \cdot w_i - 2w_{f(i)} + 2w_{i'}$.

Proof. Let i_1 be the job interrupted by i. By observation, before i_1 is either a job completion or an interruption of type $[A1]$. Then, by Lemma 1, the charge that i receives from jobs started by ADV before S_{i_1} is at most $2w_{i_1} - 2w_{f(i)}$. Due to the fact that all jobs have the same length, there are at most two jobs j_1 and j started by ADV in intervals $[S_{i_1}, S_i)$ and $[S_i, C_i)$, respectively (Figure 2). As i interrupts i_1 according to $[A2]$, at time C_i job i_1 is still pending. So at that moment, ALG will schedule some job i' with weight $w_{i'} \geq w_{i_1}$. Moreover, $d_i < S_i + 2p$ so either i receives no self-charge or $j = i$. Hence, the charge that i receives is at most $2w_{i_1} - 2w_{f(i)} + w_{j_1} + \max\{w_i, w_j\}$. In the following, we prove $2w_{i_1} + w_{j_1} + \max\{w_i, w_j\} \leq \max\left\{\frac{6}{\beta} - 1, 4\right\} \cdot w_i + 2w_{i'}$.

We have $w_{j_1} < 2w_{i_1}$ since otherwise, job j_1 would have interrupted job i_1 by condition $[A1]$. If $w_j \leq 2w_i$ then $2w_{i_1} + w_{j_1} + \max\{w_i, w_j\} \leq 2w_{i_1} + 2w_{i_1} + 2w_i \leq 4w_i + 2w_{i'}$. Consider the case $w_j > 2w_i$. Let $\tau \in [S_i, C_i)$ be the moment that ADV starts j. If j is not urgent at τ then $w_{i'} \geq w_j$, so $2w_{i_1} + w_{j_1} + w_j < 4w_i + 2w_{i'}$. In the remaining, $w_j > 2w_i$ and j is urgent at time τ.

1. **There exists job ℓ such that $S_i + 2p \leq d_\ell < \tau + 2p$ and $w_\ell \geq w_i$.** In this case, we have $w_{i'} \geq w_\ell \geq w_i$. If $w_{j_1} \leq w_i$ then $w_{i'} \geq w_{j_1}$. If $w_{j_1} > w_i$ then j_1 is not urgent at its starting time by the ADV since otherwise, j_1 would have interrupted i_1 according to $[A2]$. Hence, j_1 is pending at S_i. By the job selection in condition $[A2]$, j_1 is not urgent at S_i. Therefore, j_1 is also pending at time C_i, so $w_{i'} \geq w_{j_1}$. In both cases, $w_{i'} \geq w_{j_1}$. Besides, as j did not interrupt i according to $[A1]$ $w_j \leq \max\{2w_i, 4w_{i_1}\}$. Thus, $w_j \leq 4w_{i_1}$. We have:

$$2w_{i_1} + w_{j_1} + w_j \leq 6w_{i_1} + w_{i'} \leq (6/\beta - 1) \cdot w_i + 2w_{i'}$$

2. **There does not exist job ℓ such that $S_i + 2p \leq d_\ell < \tau + 2p$ and $w_\ell \geq w_i$.** As j is urgent, $w_j < 2w_i + w_{i_1}$ since otherwise j would have interrupted job i by $[A3]$. Hence,

$$2w_{i_1} + w_{j_1} + w_j < 2w_{i_1} + w_{j_1} + 2w_i + w_{i_1}$$

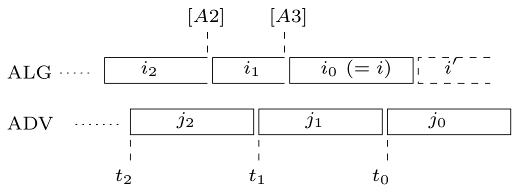


Fig. 3. Illustration of the phase with one interruption $[A3]$

If $d_{j_1} \geq S_i + 2p$ then $w_{i'} \geq \max\{w_{i_1}, w_{j_1}\}$, so $2w_{i_1} + w_{j_1} + 2w_i + w_{i_1} < 4w_i + 2w_{i'}$ (note that $w_i > w_{i_1}$). If $S_i + p < d_{j_1} < S_i + 2p$ and as j_1 is not completed before by ALG then by the choice of scheduled job at interruption $[A2]$, $w_{j_1} \leq w_i$. If $d_{j_1} < S_i + p$ then we also have $w_{j_1} \leq w_i$ since otherwise, j_1 would have interrupted i_1 according to $[A2]$. Therefore, we also have $2w_{i_1} + w_{j_1} + 2w_i + w_{i_1} \leq 4w_i + 2w_{i'}$. □

Lemma 4. *If the phase of job i is of type 3 then the charge that i receives is at most* $\max\left\{\frac{3\beta+11}{2\beta+1}, 4 + \frac{3-\beta}{5\beta+2}\right\} w_i - 2w_{f(i)} + 2w_{i'}$.

Proof. By observation, there are at most two consecutive interruptions $[A3]$ in the end of job i's phase.

*1. **There is only one** $A3$**-interruption.*** Let $i_0 (= i), i_1, i_2$ be jobs started by ALG in the phase such that i_0, i_1 interrupt i_1, i_2 according to $[A3]$ and $[A2]$, respectively. Let j_0, j_1 and j_2 be jobs started by ADV in $[S_{i_0}, C_{i_0})$, $[S_{i_1}, S_{i_0})$ and $[S_{i_2}, S_{i_1})$, respectively (Figure 3). By Lemma 1, the charge of jobs started by ADV from the beginning of the phase to S_{i_2} is at most $2w_{i_2} - 2w_{f(i_0)}$. As job i_0 is new released and urgent at time S_{i_0}, i_0 receives no self-charge or $j_0 = i_0$. In both cases, the total charge is bounded by $2w_{i_2} - 2w_{f(i_0)} + w_{j_2} + w_{j_1} + \max\{w_{j_0}, 2w_{i_0}\}$. In the following, we argue that $W := 2w_{i_2} + w_{j_2} + w_{j_1} + \max\{w_{j_0}, 2w_{i_0}\} \leq \max\left\{\frac{3\beta+11}{2\beta+1}, 4\right\} w_{i_0} + 2w_{i'}$.
Remark that j_0 does not interrupt i_0 according to $[A1]$, so $\max\{w_{j_0}, 2w_{i_0}\} \leq \max\{8w_{i_2}, 4w_{i_1}, 2w_{i_0}\} \leq \max\{2, 8/(2\beta+1)\} w_{i_0} \leq 8/(2\beta+1) w_{i_0}$ (because $\beta < 3/2$). Note that $\max\{8w_{i_2}, 4w_{i_1}, 2w_{i_0}\} \leq 2w_{i_0} + 2w_{i_2}$, so sometimes, we only need a weaker inequality $\max\{w_{j_0}, 2w_{i_0}\} \leq 2w_{i_0} + 2w_{i_2}$. We also use inequalities $w_{j_2} < 2w_{i_2}$ (because j_2 does not interrupt i_2 according to $[A1]$), and $(2\beta+1)w_{i_2} \leq w_{i_0}$.

(a) **Case $w_{j_1} > w_{i_0}$.** By condition $[A3]$, we have either $d_{j_1} \geq S_{i_0} + 2p$ or $d_{j_1} < S_{i_1} + 2p$ (otherwise, j_1 would have played the role of job ℓ at time $t = S_{i_0}$ in the definition of $[A3]$). In the former, $w_{i'} \geq w_{j_1}$. Hence,

$$4w_{i_0} + 2w_{i'} > 2w_{i_0} + 2(2w_{i_1} + w_{i_2}) + w_{j_1} > \\ > (2w_{i_0} + 2w_{i_2}) + w_{j_1} + 2w_{i_2} + 2w_{i_2} > W.$$

In the latter, job j_1 is urgent at its starting time t_1 by the ADV. Moreover, we know that no job ℓ satisfying $S_{i_1} + 2p \leq d_\ell < S_{i_0} + 2p$ and $w_\ell \geq w_{i_1}$ by definition of $[A3]$. So there is no job ℓ satisfying $S_{i_1} + 2p \leq d_\ell < t_1 + 2p$ and $w_\ell \geq w_{i_1}$. As $w_{j_1} > w_{i_0}$, j_1 would have interrupted i_1 according to condition $[A3]$ (contradiction).

(b) **Case** $w_{j_1} \leq w_{i_0}$ **and** $w_{j_2} > w_{i_1}$**.** Then j_2 is not urgent at its starting time t_2 by the ADV (since otherwise, j_2 would have interrupted i_2). By job selection in $[A2]$, j_2 is not urgent at S_{i_1} neither, i.e. $d_{j_2} \geq S_{i_1} + 2p$. Furthermore, by definition of condition $[A3]$, we have $d_{j_2} \geq S_{i_0} + 2p$ since otherwise j_2 would played the role of job ℓ in the definition of $[A3]$. Hence, $w_{i'} \geq w_{j_2} > w_{i_1} > w_{i_2}$. Therefore,

$$\begin{aligned} 4w_{i_0} + 2w_{i'} &> w_{j_1} + 2w_{i_0} + (2w_{i_1} + w_{i_2}) + w_{j_2} + w_{i_2} > \\ &> w_{j_1} + (2w_{i_0} + 2w_{i_2}) + w_{j_2} + 2w_{i_2} \geq W \end{aligned}$$

(c) **Case** $w_{j_1} \leq w_{i_0}$ **and** $w_{j_2} \leq w_{i_1}$**.** We have:

$$\begin{aligned} W = w_{j_1} + w_{j_2} + w_{j_0} + 2w_{i_2} &< w_{i_0} + w_{i_1} + \frac{8}{2\beta+1} w_{i_0} + 2w_{i_2} < \\ &< w_{i_0} + (w_{i_1} + \frac{1}{2} w_{i_2}) + \frac{8}{2\beta+1} w_{i_0} + \frac{3}{2(2\beta+1)} w_{i_0} \leq \frac{3\beta+11}{2\beta+1} \cdot w_{i_0} \end{aligned}$$

2. ***There are two*** $A3$***-interruptions.*** This analysis is similar to the previous one.

Let $i_0 (= i), i_1, i_2, i_3$ be jobs started by ALG in the phase such that i_0, i_1, i_2 interrupt i_1, i_2, i_3 according to $[A3], [A3]$ and $[A2]$, respectively. Let j_0, j_1, j_2 and j_3 be jobs started by ADV in $[S_{i_0}, C_{i_0}), [S_{i_1}, S_{i_0}), [S_{i_2}, S_{i_1})$ and $[S_{i_3}, S_{i_2})$, respectively (Figure 4). By Lemma 1, the charge of jobs started by ADV from the beginning of the phase to S_{i_3} is at most $2w_{i_3} - 2w_{f(i)}$. As job i_0 is new released and urgent at time S_{i_0}, either i_0 receives no self-charge or if i_0 receives a self-charge then $w_{j_0} < w_{i_0}$. In the latter, the total charge of i_0 is at most $2w_{i_2} - 2w_{f(i)} + w_{j_3} + w_{j_2} + w_{j_1} + w_{j_0} + w_{i_0} < 2w_{i_2} - 2w_{f(i)} + w_{j_2} + w_{j_1} + 2w_{i_0}$. In the former, i_0 receives at most $2w_{i_2} - 2w_{f(i)} + w_{j_3} + w_{j_2} + w_{j_1} + w_{j_0}$. In the following, we argue that $2w_{i_3} + w_{j_3} + w_{j_2} + w_{j_1} + \max\{w_{j_0}, 2w_{i_0}\} \leq \max\left\{\frac{3\beta+11}{2\beta+1}, 4 + \frac{3-\beta}{5\beta+2}\right\} w_{i_0} + 2w_{i'}$.

Remark that j_0 does not interrupt i_0 according to $[A1]$, so $\max\{w_{j_0}, 2w_{i_0}\} \leq \max\{2w_{i_0}, 4w_{i_1}, 8w_{i_2}, 16w_{i_3}\} \leq 2w_{i_0} + 2w_{i_2}$. Similarly, $w_{j_2} \leq \max\{2w_{i_2}, 4w_{i_3}\}$ and $w_{j_3} \leq 2w_{i_3}$.

(a) **Case** $w_{j_1} > w_{i_0}$**.** Then by condition $[A3]$, we have either $d_{j_1} \geq S_{i_0} + 2p$ or $d_{j_1} < S_{i_1} + 2p$. In the former, $w_{i'} \geq w_{j_1}$. Hence,

$$\begin{aligned} 4w_{i_0} + 2w_{i'} &> 2w_{i_0} + w_{j_1} + 3(2w_{i_1} + w_{i_2}) \\ &> (2w_{i_0} + 2w_{i_2}) + w_{j_1} + (2w_{i_2} + 2w_{i_3}) + 2w_{i_3} + 2w_{i_3} \\ &> w_{j_0} + w_{j_1} + w_{j_2} + w_{j_3} + 2w_{i_3} \end{aligned}$$

In the latter, job j_1 is urgent at its starting time t_1 by the ADV. Moreover, we know that no job ℓ satisfying $S_{i_1} + 2p \leq d_\ell < S_{i_0} + 2p$ and $w_\ell \geq w_{i_1}$ by

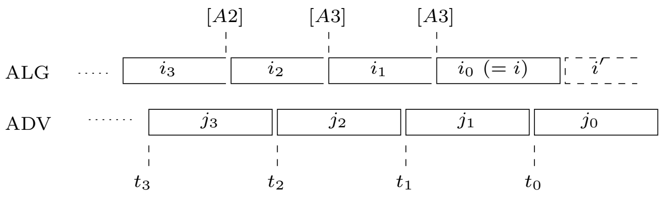


Fig. 4. Illustration for the phase with two interruptions [$A3$]

definition of [$A3$]. So there is no job ℓ satisfying $S_{i_1} + 2p \le d_\ell < t_1 + 2p$ and $w_\ell \ge w_{i_1}$. As $w_{j_1} > w_{i_0}$, j_1 would have interrupted i_1 according to condition [$A3$] (contradiction).

(b) **Case $w_{j_1} \le w_{i_0}$ and $w_{j_2} > w_{i_1}$.** Then j_2 is not urgent at its starting time t_2 by the ADV since otherwise, j_2 would have preempted i_2 according to [$A3$]. By [$A2$], $d_{j_2} \ge S_{i_1} + 2p$. Furthermore, by definition of condition [$A3$], we have $d_{j_2} \ge S_{i_0} + 2p$. Hence, $w_{i'} \ge w_{j_2} > w_{i_1}$. Therefore,

$$\begin{aligned}4w_{i_0} + 2w_{i'} &> 2w_{i_0} + w_{j_1} + w_{i_0} + w_{j_2} + w_{i_1}\\ &> (2w_{i_0} + 2w_{i_2}) + w_{j_1} + w_{j_2} + w_{i_0} + w_{i_3}\\ &> w_{j_0} + w_{j_1} + w_{j_2} + w_{j_3} + 2w_{i_3}\end{aligned}$$

(c) **Case $w_{j_1} \le w_{i_0}$ and $w_{j_2} \le w_{i_1}$.** We have:

$$\begin{aligned}w_{j_0} + w_{j_1} + w_{j_2} + w_{j_3} + 2w_{i_3} &\le (2w_{i_0} + 2w_{i_2}) + w_{i_0} + w_{i_1} + 4w_{i_3}\\ &\le 3w_{i_0} + 2w_{i_1} + 3w_{i_3} \le 4w_{i_0} + (3-\beta)w_{i_3}\\ &\le \left(4 + \frac{3-\beta}{5\beta+2}\right) w_{i_0}\end{aligned}$$

where the last inequality is due to $w_{i_3} \le \frac{1}{5\beta+2} w_{i_0}$ (by combining $w_{i_0} \ge 2w_{i_1} + w_{i_2}$, $w_{i_1} \ge 2w_{i_2} + w_{i_3}$ and $w_{i_2} \ge \beta w_{i_3}$). □

Theorem 1. *The algorithm is* $(2+\sqrt{5})(\approx 4.24)$*-competitive while* $\beta = \sqrt{5} - 1$.

Proof: Due to previous lemmas, a job i, which is completed by ALG, receives a charge at most $r = \max\{4 + \frac{3-\beta}{5\beta+2}, 3+\beta, \frac{6}{\beta} - 1, \frac{3\beta+11}{2\beta+1}\}$. The ratio r attains minimum value $(2+\sqrt{5})$ while $\beta = \sqrt{5} - 1$. □

4 A 4.24-Competitive Algorithm for Preemption with Resume

The algorithm is similar to algorithm A but the conditions depend on the remaining processing time of jobs rather than their entire processing time.

The algorithm B. Initially, set $Q := \emptyset$ and $\alpha := 0$. At time t, if there is either no currently scheduled job or a job completion, then schedule the heaviest job. Otherwise, let j be the currently scheduled job. If there is no new released job, then continue to schedule j. Otherwise, let i be a new released job with heaviest weight. Job j is interrupted if one of the following conditions holds.

$B1$. **if:** $w_i \geq 2w_j$ and $w_i \geq 2^{\alpha} w(Q)$.
do: Schedule job i. Set $\alpha := 0$ and $Q = \emptyset$.

$B2$. **if:** $\alpha = 0$, $\beta w_j \leq w_i \leq 2w_j$, i is urgent and j can be scheduled later, i.e. $d_j \geq t + p + q_j$,
do: Schedule job i^* which is the heaviest among all urgent jobs, i.e., $i^* = \arg\max\{w_\ell : d_\ell \leq t + p + q_\ell\}$. Set $\alpha = 1$ and $Q := \{j\}$.

$B3$. **if:** i is urgent, $w_i \geq 2w_j + w_{j'}$ where j' is the job previously preempted by j and there is no another job ℓ satisfying the following conditions:

$$S_j + p + q_\ell(t) \leq d_\ell < t + p + q_\ell(t) \quad \text{and} \quad w_\ell \geq w_j.$$

do: Schedule job i.

At any interruption, if $\alpha \geq 1$ then $\alpha := \alpha + 1$. If a job is completed then set $\alpha = 0$ and $Q = \emptyset$. Conventionally, if an interruption satisfies both conditions [$B1$] and [$B3$] then we refer that interruption to [$B1$].

Analysis. The charging scheme in the model of restart does not carry over since now the adversary may schedule jobs in different pieces. We present a more subtle charging scheme in order to analyze the algorithm by exploiting the equal-length property of jobs. Note that the algorithm considers the remaining processing time of jobs. Hence, at some points, the picture looks like the model in which jobs have arbitrary length. However, initially all jobs have the same length. This property is used in proving Lemma 5, the main lemma which is not valid if jobs have arbitrary length.

The Charging Scheme. Again we renumber the jobs completed by the algorithm from 1 to n, such that the completion times are ordered $0 = C_0 < C_1 < \ldots < C_n$. For every $i = 1, \ldots, n$ we divide $[C_{i-1}, C_i)$ further into intervals: Let $a = \lceil (C_i - C_{i-1})/p \rceil$. The first interval is $[C_{i-1}, C_i - (a-1)p)$. The remaining intervals are $[C_i - (b+1)p, C_i - bp)$ for every $b = a-2, \ldots, 0$. We label every interval I with a pair (b, i) such that $I = [s, C_i - bp)$ for $s = \max\{C_{i-1}, C_i - (b+1)p\}$.

The charging also consists of three steps in which the first and the last steps are the same as the ones of the charging scheme in Section 3. The second step will be done by the following procedure, which maintains for every interval $[s, t)$ a set of jobs P that are started before t by the adversary and that are not yet charged to some job of the algorithm.

```
Initially $P = \emptyset$.
**For** all intervals $[s, t)$ as defined above in left to right order, **do**
  – Let $(b, i)$ be the label of the interval.
  – Add to $P$ all jobs $j$ started by the adversary in $[s, t)$.
```

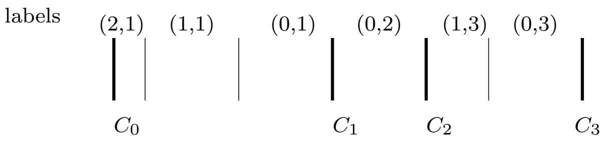


Fig. 5. The intervals as used by the charging procedure

- If P is not empty, then remove from P the job j with the smallest deadline and charge it to i. Mark $[s, t)$ with j.
- If P is empty, then leave $[s, t)$ unmarked.
- Denote by P_t the current content of P.

Note that, no job i receives any charge from job started after C_i except self-charge. Using the fact that all jobs have the same length, we prove the following main lemma.

Lemma 5. *For every interval $[s, t)$, all jobs $j \in P_t$ are still pending for the algorithm at time t.*

Proof: Assume that P_t is not empty, and let j be the job in P_t with the smallest deadline. First we claim that there is a time s_0, such that every interval contained in $[s_0, t)$ is marked with some job j' satisfying $s_0 \le r_{j'}$ and $d_{j'} \le d_j$.

Let $[s', t')$ be the interval where the adversary started j. So j entered P by the charging procedure at that interval. Job j was in P during all the iterations until $[s, t)$, so every interval between s' and t is marked with some job of deadline at most d_j. Let $\mathcal{M}$ be the set of these jobs. If for every $j' \in \mathcal{M}$ we have $s' \le r_{j'}$, choose $s_0 = s'$ and we are done. Otherwise let $j' \in \mathcal{M}$ be the job with smallest release time. So $r_{j'} < s'$. Let $[s'', t'')$ be the interval where the adversary started j'. By the same argment as above, during the iteration over the intervals between s'' and s', job j' was in P. Therefore every such interval was marked with some job with deadline at most $d_{j'} \le d_j$. Now we repeat for s'' the argument we had for s'. Eventually we obtain a valid s_0, since P was initially empty. That proves the claim.

Now let $\mathcal{M}$ be the set of jobs charged during all intervals in $[s_0, t)$. So $j \notin \mathcal{M}$. In an Earliest Deadline First schedule of the adversary, job j would be completed not before $s_0 + (|\mathcal{M}| + 1)p$. But any interval has size at most p, so $t - s_0 \le |\mathcal{M}|p$. We conclude that $d_j \ge t + p$, which shows that j is still pending for the algorithm at time t. □

Using Lemma 5, the remaining analysis follows the same structure as in the model of restart, though with different details. The lemmas and their proofs could be found in the full version.

Theorem 2. *The algorithm B is $(2+\sqrt{5})(\approx 4.24)$-competitive when $\beta = \sqrt{5}-1$.*

References

1. Baruah, S.K., Haritsa, J., Sharma, N.: On-line scheduling to maximize task completions. In: Real-Time Systems Symposium, pp. 228–236 (December 1994)
2. Chan, W.-T., Lam, T.W., Ting, H.-F., Wong, P.W.H.: New results on on-demand broadcasting with deadline via job scheduling with cancellation. In: Proc. 10th International on Computing and Combinatorics Conference, pp. 210–218 (2004)
3. Chen, B., Potts, C.N., Woeginger, G.J.: A review of machine scheduling: Complexity, Algorithms and Approximability. In: Handbook of Combinatorial Optimization, vol. 3, pp. 21–169. Kluwer Academic Publishers, Dordrecht (1998)
4. Chin, F.Y.L., Fung, S.P.Y.: Online scheduling with partial job values: Does timesharing or randomization help? Algorithmica 37(3), 149–164 (2003)
5. Chrobak, M., Jawor, W., Sgall, J., Tichý, T.: Online scheduling of equal-length jobs: Randomization and restarts help. SIAM J. Comput. 36(6), 1709–1728 (2007)
6. Dürr, C., Jeż, Ł., Nguyen, K.T.: Online scheduling of bounded length jobs to maximize throughput. In: Bampis, E., Jansen, K. (eds.) WAOA 2009. LNCS, vol. 5893, pp. 116–127. Springer, Heidelberg (2010)
7. Englert, M., Westermann, M.: Considering suppressed packets improves buffer management in QoS switches. In: SODA 2007, pp. 209–218. ACM/SIAM (2007)
8. Goldwasser, M.H.: A survey of buffer management policies for packet switches. SIGACT News 41(1) (2010)
9. Kim, J.-H., Chwa, K.-Y.: Scheduling broadcasts with deadlines. Theoretical Computer Science 325(3), 479–488 (2004)
10. Koo, C.-Y., Lam, T.-W., Ngan, T.-W., Sadakane, K., To, K.-K.: On-line scheduling with tight deadlines. Theoretical Computer Science 295(1-3), 251–261 (2003)
11. Leung, J., Kelly, L., Anderson, J.H.: Handbook of Scheduling: Algorithms, Models, and Performance Analysis. CRC Press, Boca Raton (2004)
12. Zheng, F., Fung, S.P.Y., Chan, W.-T., Chin, F.Y.L., Poon, C.K., Wong, P.W.H.: Improved on-line broadcast scheduling with deadlines. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 320–329. Springer, Heidelberg (2006)

Recognizing Sparse Perfect Elimination Bipartite Graphs

Matthijs Bomhoff

Faculty of Electrical Engineering, Mathematics and Computer Science
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
m.j.bomhoff@utwente.nl

Abstract. When applying Gaussian elimination to a sparse matrix, it is desirable to avoid turning zeros into nonzeros to preserve sparsity. Perfect elimination bipartite graphs are closely related to square matrices that Gaussian elimination can be applied to without turning any zero into a nonzero. Existing literature on the recognition of these graphs mainly focuses on time complexity. For $n \times n$ matrices with m nonzero elements, the best known algorithm runs in time $O\left(n^3/\log n\right)$. However, the space complexity also deserves attention: it may not be worthwhile to look for suitable pivots for a sparse matrix if this requires $\Omega\left(n^2\right)$ space. We present two new recognition algorithms for sparse instances: one with a $O(nm)$ time complexity in $\Theta\left(n^2\right)$ space and one with a $O\left(m^2\right)$ time complexity in $\Theta(m)$ space. Furthermore, if we allow only pivots on the diagonal, our second algorithm is easily adapted to run in time $O(nm)$.

Keywords: perfect elimination, algorithms, bipartite graphs, sparse graphs.

1 Introduction

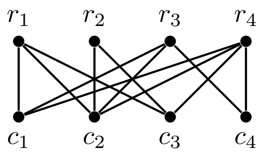
Performing Gaussian elimination on sparse matrices may have the unfortunate side effect of turning zeros into nonzero values (*fill-in*), possibly even leading to a dense matrix along the way. Clearly, this can be undesirable, for example when working with very large sparse matrices. A natural question therefore is to ask when we can avoid fill-in during the elimination process. Recognizing matrices where fill-in can be avoided and selecting appropriate pivots can decrease the required effort and space for Gaussian elimination. For several special cases, such as symmetric (positive definite) matrices or pivots chosen along the main diagonal, this problem has been treated extensively in literature (see e.g. [1–5]).

The general case of avoiding fill-in on square nonsingular matrices was first treated in detail by Golumbic and Goss [6]. They describe the correspondence between matrices that allow Gaussian elimination without fill-in and bipartite graphs. Under the assumption that subtracting a multiple of a row from another will always turn at most one element from nonzero to zero, an instance of the problem can be represented by a $\{0,1\}$ matrix M where $M_{i,j} = 1$ denotes that the original matrix contains a nonzero value at element (i,j). Given such an $n \times n$

square matrix M with m nonzero elements, we can construct the bipartite graph $G[M]$ with vertices corresponding to the rows and columns in M where vertices i and j are adjacent iff $M_{i,j}$ is nonzero. We assume each row and each column of M contains at least one nonzero element, so $G[M]$ contains no isolated vertices. For example, the matrix shown in Fig. 1(a) corresponds to the bipartite graph shown in Fig. 1(b). Golumbic and Goss called the class of bipartite graphs corresponding to matrices that allow Gaussian elimination without fill-in *perfect elimination bipartite graphs.* This class is characterized using an elimination scheme detailed in the next section. Based on this scheme, they also obtained a first algorithm for the recognition of this class. Improved algorithms for the recognition of this class of graphs and their associated matrices have subsequently been published and are discussed briefly in what follows.

$$\begin{pmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

(a)



(b)

Fig. 1. Example $\{0,1\}$-matrix M and its bipartite graph $G[M]$

The correspondence between perfect elimination bipartite graphs and matrices is mainly of practical value for sparse instances: the original motivation for investigating this class of graphs is preserving sparsity during Gaussian elimination on their associated matrices by avoiding fill-in. For the specific case of pivots chosen on the diagonal, Rose and Tarjan [5] have described two algorithms for finding perfect elimination orderings. Their algorithms represent the common trade-off between time and space. One is faster but needs more space, the other is slower but requires storage proportional to the number of nonzero elements. However, for the general case it appears that efficient algorithms for the recognition of sparse instances have not yet been investigated. The focus in literature so far seems to be only on time complexity for dense instances. The best known algorithms for the general case are based on a matrix multiplication which may well result in a dense matrix, see e.g. Fig. 2.

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 4 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Fig. 2. Sparse M may lead to dense $Q = MM^T$

New results. In this paper, we present two algorithms for efficient recognition of sparse instances (where 'sparse' is used to indicate $m \ll n^2$): one with a $O(nm)$ time complexity in $\Theta\left(n^2\right)$ space and one with a $O\left(m^2\right)$ time complexity in $\Theta(m)$ space. We also show how our second algorithm can be adapted to solve the problem in time $O(nm)$ if only pivots on the diagonal are allowed.

The remainder of this paper is organized as follows: The next section describes the class of perfect elimination bipartite graphs as well as existing literature on algorithms for its recognition. The third section describes a new version of the algorithm by Goh and Rotem for the recognition of perfect elimination bipartite graphs that has been adapted to achieve a time complexity of $O(nm)$ instead of $O\left(n^3\right)$. The section after that describes a new recognition algorithm with a time complexity of $O\left(m^2\right)$ and a space complexity of $\Theta(m)$. Finally, we present a discussion on other possible improvements as well as a few brief conclusions regarding our results.

2 Perfect Elimination Bipartite Graphs

An edge uv of a bipartite graph is called *bisimplicial* if the neighbors of its endpoints $\Gamma(u) \cup \Gamma(v)$ (where $\Gamma(u)$ denotes the neighbors of u) induce a complete bipartite graph. Bisimplicial edges in $G[M]$ correspond to pivots that avoid fill-in in M. Using this notion, perfect elimination bipartite graphs were first defined by Golumbic and Goss [6] as follows:

Definition 1. *A bipartite graph $G = (U, V, E)$ is called* perfect elimination bipartite*, if there exists a sequence of pairwise nonadjacent edges $[u_1v_1, \ldots, u_nv_n]$ such that u_iv_i is a bisimplicial edge of $G - \{u_1, v_1, \ldots, u_{i-1}, v_{i-1}\}$ for each i and $G - \{u_1, v_1, \ldots, u_n, v_n\}$ is empty. Such a sequence of edges is called a* (perfect elimination) scheme*.*

This definition is based on the following theorem:

Theorem 1. *If uv is a bisimplicial edge of a perfect elimination bipartite graph $G = (U, V, E)$, then $G - \{u, v\}$ is also a perfect elimination bipartite graph.*

This theorem immediately implies a simple $O\left(n^5\right)$ algorithm for the recognition of perfect elimination bipartite graphs that also leads to an elimination scheme in case the graph is perfect elimination bipartite. Let us introduce the notion of row and column sets R_i and C_j defined as follows:

$$R_i = \{j \in \{1 \ldots n\} | M_{i,j} \neq 0\} \tag{1}$$

$$C_j = \{i \in \{1 \ldots n\} | M_{i,j} \neq 0\} \tag{2}$$

In other words: R_i contains the column numbers of elements in row i that have a nonzero value in M. Using these, we can describe the algorithm by Golumbic and Goss, shown in Algorithm 1. The algorithm basically performs n iterations, during each of which all remaining edges are completely checked for bisimpliciality.

Algorithm 1. Original recognition algorithm by Golumbic and Goss

```
1: $I \leftarrow \{1 \dots n\}$
2: $J \leftarrow \{1 \dots n\}$
3: while $I \neq \emptyset$ do
4:    $f \leftarrow$ false
5:    for all $(i,j) \in I \times J$ do
6:       if $M_{i,j} = 1$ then
7:          $g \leftarrow$ true
8:          for all $(k,l) \in (C_j \cap I) \times (R_i \cap J)$ do
9:             if $M_{k,l} = 0$ then
10:               $g \leftarrow$ false
11:         if $g =$ true then
12:            $f =$ true, $x \leftarrow i, y \leftarrow j$
13:   if $f =$ false then
14:      return false {$G[M]$ is not perfect elimination bipartite}
15:   $I \leftarrow I \setminus x$
16:   $J \leftarrow J \setminus y$
17: return true {$G[M]$ is perfect elimination bipartite}
```

Goh and Rotem [7] have presented a faster recognition algorithm based on the following: A row $M_{a,*}$ is said to *majorize* a row $M_{b,*}$ if for each $1 \leq j \leq n$ we have $M_{a,j} \geq M_{b,j}$. According to this definition, every row majorizes itself.

Theorem 2. *[7] Let M be an $n \times n$ $\{0,1\}$ matrix representing a bipartite graph $G = (U, V, E)$. Let ℓ_i be the number of rows in M that majorize row i and let s_j be the sum of the entries in column j of M. Then $M_{i,j} = 1$ and $\ell_i = s_j$ iff the edge $u_i v_j$ is a bisimplicial edge of G.*

The values ℓ_i can be easily determined using the matrix $Q = MM^T$: ℓ_i is equal to the number of elements in the row $Q_{i,*}$ that are equal to $Q_{i,i}$ (including $Q_{i,i}$ itself). Once the matrix Q is computed, finding a bisimplicial edge can be done in $O\left(n^2\right)$ operations. If a bisimplicial edge is found, Q can be updated in $O\left(n^2\right)$ operations to the matrix Q' associated with $G' = G - \{u, v\}$ for the next iteration. After at most n iterations, the algorithm terminates, so the total time complexity of the algorithm is $O\left(n^3\right)$, a significant improvement over the $O\left(n^5\right)$ naive implementation. This algorithm is shown in Algorithm 2 (The notation M^{ij} is used to denote the (i,j) minor of M). As it needs to compute and store the matrix Q, its space complexity is $\Theta\left(n^2\right)$.

More recently, Spinrad [8] obtained an improved algorithm with time complexity $O\left(n^3 / \log n\right)$ using a notion of edges that may soon become suitable pivots during subsequent iterations as well as the faster matrix multiplication algorithm by Coppersmith and Winograd [9].

3 Goh-Rotem on Sparse Instances

By adapting the way calculations are performed, as well as the data structures, we obtain a new implementation of Algorithm 2 with time complexity $O(nm)$:

Algorithm 2. Recognition algorithm by Goh and Rotem

```
1: simplicial_found ← **true**
2: compute the matrix $Q = (Q_{i,j})$ where $Q = MM^T$
3: $\forall j \in \{1 \dots n\} : s_j \leftarrow \sum_{i=1}^{n} M_{i,j}$
4: while there exists an $s_j \neq 0$ and simplicial_found do
5:     $\forall i \in \{1 \dots n\}$ : let $\ell_i$ be the number of entries in row $i$ of $Q$ which are equal to
       $Q_{i,i}$
6:     if there exists a nonzero entry $M_{i,j}$ in $M$ where $s_j = \ell_i$ then
7:         Compute the matrix $D = (d_{k,l})$ where $d_{k,l} = M_{k,j} \cdot M_{l,j}$
8:         $Q \leftarrow (Q - D)^{ii}$ {$Q$ is now equal to $(M^{ij})(M^{ij})^T$}
9:         $\forall k \in \{1 \dots n\} : s_k \leftarrow s_k - M_{i,k}$
10:        $s_j \leftarrow 0$
11:    else
12:        simplicial_found ← false
13: return simplicial_found
```

an improvement for sparse graphs. Using the row and column sets we determine the matrix $Q = MM^T$ as

$$Q_{i,j} = |R_i \cap R_j| \ . \tag{3}$$

Based on this new formulation, we arrive at the following lemma that will be used below to derive the time complexity of our new algorithm:

Lemma 1. *An upper bound on the sum of the elements in Q is given by*

$$\sum_{i,j} Q_{i,j} \leq nm \ . \tag{4}$$

Proof.

$$\sum_{i,j} Q_{i,j} = \sum_{i,j} |R_i \cap R_j| \leq \sum_i \sum_j |R_j| = nm \tag{5}$$

□

Besides the matrix Q, we require an additional $n \times (n+1)$ matrix B. To simplify notation, we number the columns of B starting at 0 instead of at 1. The values of B are defined by

$$B_{i,k} := |\{j | j \in 1 \dots n, Q_{i,j} = k\}| \ . \tag{6}$$

I.e., $B_{i,k}$ contains the number of elements in row i of Q that have the value k. After computation of Q, the matrix B can be computed in time $O\left(n^2\right)$. Furthermore, without increasing the time complexity of an algorithm, we can keep B up to date if we perform any updates to elements of Q. Using B, we can easily determine the value of ℓ_i as

$$\ell_i = B_{i,Q_{i,i}} \ . \tag{7}$$

Using our set-based calculation of Q and the new matrix B, we can adapt the original algorithm by Goh and Rotem and arrive at our new version shown in Algorithm 3. Apart from our use of the sets I and J to denote the rows and columns that are still part of M during the current iteration instead of taking minors of the involved matrices, the working of the algorithm is still basically identical to Algorithm 2. However, the additional bookkeeping of B and the upper bound on the sum of the elements in Q enable us to achieve an improved time complexity for sparse instances.

Algorithm 3. Adapted Goh-Rotem algorithm

```
Require: $Q = 0$ {$Q$ is a $n \times n$-matrix}
Require: $B = 0$ {$B$ is a $n \times (n+1)$-matrix}
 1: $I \leftarrow \{1 \dots n\}$
 2: $J \leftarrow \{1 \dots n\}$
 3: for all $(i,j) \in I \times J$ do
 4:     $Q_{i,j} \leftarrow |R_i \cap R_j|$
 5:     $B_{i,Q_{i,j}} \leftarrow B_{i,Q_{i,j}} + 1$
 6: $\forall j \in J : s_j \leftarrow |C_j|$
 7: while $I \neq \emptyset$ do
 8:     $f \leftarrow$ false
 9:     for all $i \in I$ do
10:         for all $j \in R_i$ do
11:             if $B_{i,Q_{i,i}} = s_j$ then
12:                 $f \leftarrow$ true, $x \leftarrow i, y \leftarrow j$
13:     if $f =$ false then
14:         return false {$G[M]$ is not perfect elimination bipartite}
15:     $\forall i \in I : R_i \leftarrow R_i \setminus y$
16:     for all $j \in J$ do
17:         if $x \in C_j$ then
18:             $s_j \leftarrow s_j - 1$
19:         $C_j \leftarrow C_j \setminus x$
20:     for all $(i,j) \in C_y \times C_y$ do
21:         $B_{i,Q_{i,j}} \leftarrow B_{i,Q_{i,j}} - 1$
22:         $Q_{i,j} \leftarrow Q_{i,j} - 1$
23:         $B_{i,Q_{i,j}} \leftarrow B_{i,Q_{i,j}} + 1$
24:     $\forall i \in I : B_{i,Q_{i,x}} \leftarrow B_{i,Q_{i,x}} - 1$
25:     $I \leftarrow I \setminus x$
26:     $J \leftarrow J \setminus y$
27: return true {$G[M]$ is perfect elimination bipartite}
```

Theorem 3. *The time complexity of Algorithm 3 is $O(nm)$.*

Proof. From Lemma 1 we know the sum of the elements of Q is bounded by $O(nm)$. This implies the initialization of the matrices Q and B in the loop on line 3 can be completed within time $O(nm)$. This leaves us with the task of establishing the same bound on the main loop from line 7 on down. Clearly, the main loop is executed up to n times, either finding and processing a pivot, or returning false during each iteration. Within the main loop, the first loop on line 9 processes each of the $O(m)$ edges in constant time. If a suitable pivot is found, we first update the R_i and C_j sets in lines 15 and 16. This can be done in time $O(m)$.

After that, we have to update the matrices Q and B in the loop on line 20. Every iteration of this inner loop decreases some element of Q by one. As none of the elements are decreased below zero, Lemma 1 again gives us a bound of $O(nm)$ on the number of iterations of this inner loop over the course of the entire algorithm.

Finally, the loop on line 24 decreases $O(n)$ values of B after which I and J are updated to reflect the removal of the pivot row and column; all of this can be done in time $O(n)$.

So for both the initialization and the iteration phase of the algorithm we found a bound of $O(nm)$ on the time complexity. □

The space complexity of Algorithm 3 is $\Theta\left(n^2\right)$ as we need to compute and store the matrices Q and B.

4 Avoiding Matrix Multiplication

A possible disadvantage of recognition algorithms based on matrix multiplication is the amount of space required to store the result of the matrix multiplication. Even if an original sparse matrix M is stored efficiently using $\Theta(m)$ space, the result of the multiplication may be a dense matrix requiring $\Theta\left(n^2\right)$ space (see Fig. 2). Avoiding matrix multiplication thus seems to be required in order to improve the space complexity. To do this, we started over from the algorithm originally presented by Golumbic and Goss for the recognition of perfect elimination bipartite graphs. Algorithm 1 proceeds in up to n iterations. In every iteration, every edge is checked against possibly all other edges to determine if it is bisimplicial. To check an edge uv for bisimplicity, we need to verify that $G[M]$ contains all edges $u'v'$ with $u' \in \Gamma(v)$ and $v' \in \Gamma(u)$. By performing this every iteration, we obtain a time complexity of $O\left(n^5\right)$.

The idea behind our new algorithm is as follows: in Algorithm 1 we check every remaining edge uv against possibly all other edges during every iteration. However, we can shave a factor n from the time complexity if we are checking uv and find an edge $u'v'$ as present in $G[M]$ during some iteration, we avoid checking it for uv again in subsequent iterations. A naive algorithm based on this notion is described in Algorithm 4. Assuming the use of suitable data structures, the time complexity of this algorithm is $O\left(n^2 m\right)$. Unfortunately, by precomputing for every edge e the set of possible edges E_e that need to be checked, we require a lot more space, instead of less.

Observing the usage of the sets E_e, we see they are all constructed at the beginning and processed one element at a time in arbitrary order. The element

Algorithm 4. A $O\left(n^2m\right)$ recognition algorithm

```
1: $I \leftarrow \{1 \dots n\}$
2: $J \leftarrow \{1 \dots n\}$
3: $\forall e = (i,j) \in E : E_e = C_j \times R_i$
4: while $I \neq \emptyset$ do
5:    $f \leftarrow$ false
6:    for all $e = (i,j) \in E$ do
7:       $g \leftarrow$ true
8:       if $i \notin I \vee j \notin J$ then
9:          $g \leftarrow$ false
10:      while $(E_e \neq \emptyset) \wedge (g =$ true$)$ do
11:         $e' = (i',j') \leftarrow$ arbitrary_element$(E_e)$
12:         if $i' \notin I \vee j' \notin J$ then
13:            $E_e \leftarrow E_e \setminus e'$
14:         else if $M_{i',j'} = 1$ then
15:            $E_e \leftarrow E_e \setminus e'$
16:         else
17:            $g \leftarrow$ false
18:      if $g =$ true then
19:         $f \leftarrow$ true$, x \leftarrow i, y \leftarrow j$
20:   if $f =$ false then
21:      return false {$G[M]$ is not perfect elimination bipartite}
22:   $I \leftarrow I \setminus x$
23:   $J \leftarrow J \setminus y$
24: return true {$G[M]$ is perfect elimination bipartite}
```

under consideration is either removed from the set and followed by another element, or it leads to the conclusion that e is not bisimplicial in the matrix that remains in the current iteration and it will be considered again later. If we impose a specific order on the processing of the edges $e \in E_e$, we can do away with precomputing and storing the entire sets E_e and only store the element e' currently under consideration for each edge e.

To implement this, we again represent M using the sets R_i and C_j, but this time we store them as sorted lists, as shown in Fig. 3(a). To perform a pivot and remove the associated row and column, we simply adjust the links in the row and column lists to skip over the removed row and column, as shown in Fig. 3(b) for a pivot on $(3,4)$. Clearly, such a pivot operation can be implemented in time $O(m)$, as we can simply pass over all the elements in each of the lists and adjust the links as we pass them. This representation requires $\Theta(m)$ space.

To check if an element $M_{i,j}$ corresponds to a bisimplicial edge in $G[M]$, we have to test if all edges between the neighbors of its endpoints exist. In terms of the column sets of the matrix M, this means that for every column $k \in R_i$, we must have that $C_j \subseteq C_k$. If we use the sorted list representation, the number of comparisons for each edge e is bounded by $O(m)$. Every comparison has one of three possible outcomes (see Fig. 4):

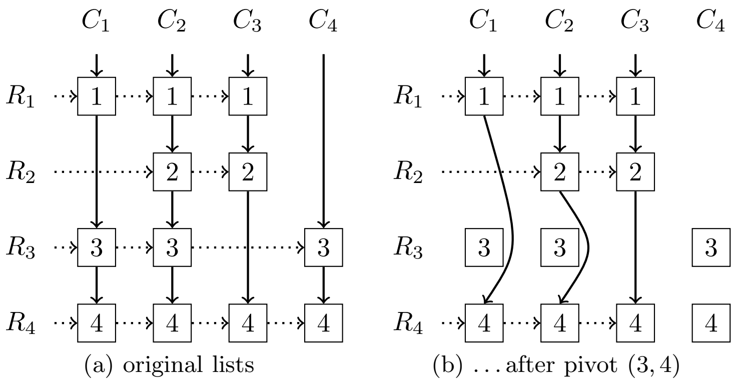


Fig. 3. Row and column lists for example matrix M from Fig. 1(a)

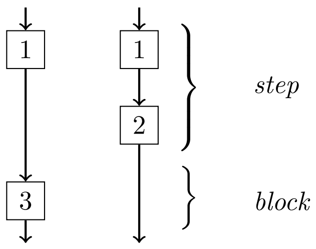


Fig. 4. Steps and blocks

1. C_j and C_k both contain the row number: the required edge is present, we can continue checking the next row number
2. C_k contains a number not present in C_j: an additional edge is present, we can continue checking the next row number
3. C_j contains a number not present in C_k: a required edge is missing: e is not bisimplicial in the current matrix

We call the first two cases 'steps' (as they can be repeated during a single iteration) and call the third case a 'block' (as it ends the checks for e during this iteration). For a single edge e, steps can occur $O(m)$ times during the algorithm, whereas blocks are limited by $O(n)$ as they can occur only once per iteration. If there are no more comparisons left for any edge e that still remains in M at some point during the algorithm, we have found a suitable pivot. After removing the pivot row and column from M, we simply proceed checking the remaining edges starting at the point where they blocked during the previous iteration. We continue this process until either we have found a complete elimination scheme or we cannot find a bisimplicial edge anymore. This procedure is described in Algorithm 5.

Algorithm 5. A new $O\left(m^2\right)$ recognition algorithm using $\Theta\left(m\right)$ space

```
1: $I \leftarrow \{1 \dots n\}$
2: $J \leftarrow \{1 \dots n\}$
3: Construct $R_i$ and $C_j$ representation
4: while $I \neq \emptyset$ do
5:    $f \leftarrow$ false
6:    for all $e = (i, j) \in E$ do
7:       $g \leftarrow$ true
8:       if $i \notin I \vee j \notin J$ then
9:          $g \leftarrow$ false
10:      while $g =$ true and we are not done checking edges do
11:         $e' = (i', j') \leftarrow$ the current edge to check
12:         if $i' \notin I \vee j' \notin J$ then
13:            Proceed to the next edge to check (if any) {This can only happen during
               the first iteration of this inner loop}
14:         else if $e'$ blocks then
15:            $g \leftarrow$ false
16:         else
17:            Proceed to the next edge to check (if any)
18:      if $g =$ true then
19:         $f \leftarrow$ true, $x \leftarrow i, y \leftarrow j$
20:   if $f =$ false then
21:      return false {$G[M]$ is not perfect elimination bipartite}
22:   Update $R_i$ and $C_j$ links to perform pivot round $(x, y)$
23:   $I \leftarrow I \setminus x$
24:   $J \leftarrow J \setminus y$
25: return true {$G[M]$ is perfect elimination bipartite}
```

Theorem 4. *The time complexity of Algorithm 5 is* $O\left(m^2\right)$.

Proof. It is possible to construct our sorted list representation in time $O\left(n^2\right)$. Other initialization, such as the state of the comparisons for every edge e, can easily be done within the same time. Following the initialization, we perform up to n iterations, during each of which we perform a pivot on our rows and columns lists in time $O\left(m\right)$ for a total of $O\left(nm\right)$. During the entire algorithm we perform $O\left(m\right)$ 'steps' and $O\left(n\right)$ 'blocks' for each of the m edges, leading to an overall time complexity of $O\left(m^2\right)$. $\square$

Theorem 5. *The space complexity of Algorithm 5 is* $\Theta\left(m\right)$.

Proof. Our sorted lists representation of M contains m edges. For each edge we need $\Theta\left(1\right)$ space to store its progress with respect to its comparisons against its required neighbors for a total of $\Theta\left(m\right)$. Finally, we have to store the sets I and J to keep track of the rows and columns that still remain, both require $\Theta\left(n\right)$ space. In total, we thus obtain a space complexity of $\Theta\left(m\right)$. $\square$

After establishing its running time and space requirements, we end this section by adapting our new algorithm to the special case of finding a perfect elimination ordering allowing only pivots on the diagonal of the matrix. Rose and Tarjan have studied this problem and have presented two algorithms for it also focusing on the trade-off between time and space requirements [5]. One of their algorithms has a time complexity of $O(nm)$ and uses $\Theta(nm)$ space, the other one has a time complexity of $O\left(n^2m\right)$ but uses only $\Theta(m)$ space.

It is not hard to see that our algorithm can be adapted to consider only a subset of all the edges as pivots: this simply means we only process steps and blocks for these edges while ignoring the other edges. If we test only c edges as allowed pivots in this way, the running time of our algorithm is $O(cm + nm)$ while the space complexity remains $\Theta(m)$. By only allowing pivots on the diagonal ($c = n$) instead of anywhere ($c = m$), we get a time complexity of $O(nm)$ for this restricted case. We thus obtain a single algorithm that combines the best time complexity of Rose and Tarjan with their best space complexity for this restricted problem.

5 Discussion

In the previous sections, we have presented two new algorithms for the recognition of perfect elimination bipartite graphs. Both are aimed at efficient recognition of sparse instances, the trade-off between the two is in the amount of space required, respectively $\Theta\left(n^2\right)$ and $\Theta(m)$.

Besides improving time and space complexity, another interesting aspect of algorithmic performance is the possibility of parallelization. From the algorithm of Goh and Rotem, it is not too hard to see that finding a single bisimplicial edge can be done in polylog time given a polynomial number of processors: matrix multiplication can be performed in polylog time [10] as well as the post-processing to determine the values of ℓ_i and check the individual matrix elements for bisimpliciality. Our new algorithm can be parallelized on $O\left(m^2\right)$ processors to find a bisimplicial edge in polylog time as all checks for all edges can be performed in parallel and subsequently combined in polylog time to find a bisimplicial edge if one exists. It is however unclear if it is also possible to use a polynomial number of processors to run the entire recognition process in polylog time: all currently known recognition algorithms are based on finding an elimination sequence of n bisimplicial edges and this appears to be an inherently sequential process. A fundamentally different approach might be necessary in order to achieve more parallelism and obtain a polylog time approach for the entire recognition process.

Another subject for further investigation is that of minimizing fill-in when it cannot be avoided completely. For symmetric positive definite matrices with pivots chosen along the main diagonal, minimizing the fill-in in the associated chordal graphs has been shown to be NP-hard [11]. Furthermore, for fill-in in chordal graphs an approximation algorithm has been developed [12]. As far as we know, the complexity of minimizing fill-in for general matrices and perfect elimination bipartite graphs is unknown. Considering the practical applications of minimum elimination orderings, obtaining results on the complexity in the general case, as well as either a polynomial time algorithm or an approximation algorithm for minimizing fill-in seem to be good topics for further research.

6 Conclusion

In current literature, the fastest known algorithm for the recognition of general perfect elimination bipartite graphs is the algorithm by Spinrad [8] with a time complexity of $O\left(n^3/\log n\right)$. We have presented two new algorithms focused specifically on sparse instances. Our first algorithm is an adaption of the algorithm by Goh and Rotem with a time complexity of $O\left(nm\right)$, leading to an improvement for instances with $m = o\left(n^2/\log n\right)$ (all but the densest instances). The second algorithm we have presented is not based on some form of matrix multiplication and is as such able to do away with the $\Omega\left(n^2\right)$ space complexity associated with it. This algorithm has a time complexity of $O\left(m^2\right)$ and a space complexity of just $\Theta\left(m\right)$. For instances with $m = o\left(n\sqrt{n\log n}\right)$ this algorithm is faster than the algorithm by Spinrad while requiring less space. We have also shown how the restricted problem where only pivots on the diagonal are allowed can be solved in time $O\left(nm\right)$ using an adapted version of our algorithm.

Interesting subjects for further study might be algorithms that parallelize better as well as problems related to the minimum fill-in on general square matrices, such as its complexity, (exact) algorithms and approximations.

Acknowledgement. The author gratefully acknowledges the support of the Innovation-Oriented Research Programme 'Integral Product Creation and Realization (IOP IPCR)' of the Netherlands Ministry of Economic Affairs, Agriculture and Innovation.

References

1. Rose, D.J.: A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In: Read, R.C. (ed.) Graph Theory and Computing, pp. 183–217. Academic Press, New York (1972)
2. Haskins, L., Rose, D.J.: Toward Characterization of Perfect Elimination Digraphs. SIAM J. Comput. 2, 217–224 (1973)
3. Kleitman, D.J.: A Note on Perfect Elimination Digraphs. SIAM J. Comput. 3, 280–282 (1974)
4. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic Aspects of Vertex Elimination on Graphs. SIAM J. Comput. 5, 266–283 (1976)
5. Rose, D.J., Tarjan, R.E.: Algorithmic Aspects of Vertex Elimination on Directed Graphs. SIAM J. Appl. Math. 34, 176–197 (1978)
6. Golumbic, M.C., Goss, C.F.: Perfect Elimination and Chordal Bipartite Graphs. J. Graph Theory 2, 155–163 (1978)
7. Goh, L., Rotem, D.: Recognition of perfect elimination bipartite graphs. Inform. Process. Lett. 15, 179–182 (1982)

8. Spinrad, J.P.: Recognizing quasi-triangulated graphs. Discrete Appl. Math. 138, 203–213 (2004)
9. Coppersmith, D., Winograd, S.: Matrix Multiplication via Arithmetic Progressions. J. Symbolic Comput. 9, 251–280 (1990)
10. Papadimitriou, C.H.: Computational Complexity. Addison-Wesley Publishing Company, Inc., Reading (1994)
11. Yannakakis, M.: Computing the Minimum Fill-In is NP-Complete. SIAM J. Alg. Disc. Meth. 2, 77–79 (1981)
12. Natanzon, A., Shamir, R., Sharan, R.: A Polynomial Approximation Algorithm for the Minimum Fill-In Problem. In: STOC 1998: Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing, pp. 41–47. ACM, New York (1998)

A Multiple-Conclusion Calculus for First-Order Gödel Logic

Arnon Avron and Ori Lahav

School of Computer Science, Tel-Aviv University, Israel
{aa,orilahav}@post.tau.ac.il

Abstract. We present a multiple-conclusion hypersequent system for the standard first-order Gödel logic. We provide a constructive, direct, and simple proof of the completeness of the cut-free part of this system, thereby proving both completeness for its standard semantics, and the admissibility of the cut rule in the full system. The results also apply to derivations from assumptions (or "non-logical axioms"), showing that such derivations can be confined to those in which cuts are made only on formulas which occur in the assumptions. Finally, the results about the multiple-conclusion system are used to show that the usual single-conclusion system for the standard first-order Gödel logic also admits (strong) cut-admissibility.

1 Introduction

In [15] Gödel introduced a sequence $\{G_n\}$ $(n \geq 2)$ of n-valued matrices in the language of propositional intuitionistic logic. He used these matrices to show some important properties of intuitionistic logic. An infinite-valued matrix G_ω in which all the $G_n s$ can be embedded was later introduced by Dummett in [13]. G_ω, in turn, can naturally be embedded in a matrix $G_{[0,1]}$, the truth-values of which are the real numbers between 0 and 1 (inclusive). It has not been difficult to show that the logics of G_ω and $G_{[0,1]}$ are identical, and both are known today as "Gödel logic".[1] Later it has been shown that this logic is also characterized as the logic of *linear* intuitionistic Kripke frames (see e.g. [14]). Gödel logic is probably the most important intermediate logic, i.e. a logic between intuitionistic logic and classical logic, which turns up in several places. Recently it has again attracted a lot of attention because of its recognition as one of the three most basic fuzzy logics [16].

Gödel logic can be naturally extended to the first-order framework. In particular, the standard first-order Gödel logic (the logic based on $[0,1]$ as the set of truth-values) has been introduced and investigated in [21] (where it was called "intuitionistic fuzzy logic"). The Kripke-style semantics of this logic is provided by the class of all linearly ordered Kripke frames with *constant domains*.

[1] It is also called Gödel-Dummett logic, because it was first introduced and axiomatized in [13]. The name Dummett himself has used is *LC*.

A cut-free Gentzen-type formulation for Gödel logic was first given by Sonobe in [18]. Since then several other such calculi which employ ordinary sequents have been proposed (see [10,1,11,5,12]). All these calculi have the drawback of using some ad-hoc rules of a nonstandard form, in which several occurrences of connectives are involved. In contrast, in [2] a cut-free Gentzen-type proof system **HG** for propositional Gödel logic was introduced, which does not have this drawback. **HG** uses (single-conclusion) *hypersequents* (a natural generalization of Gentzen's original (single-conclusion) sequents), and it has exactly the same logical rules as the usual Gentzen-type system for propositional intuitionistic logic. **HG** was furthermore extended by Baaz, Ciabattoni, Fermüller, and Zach to provide appropriate proof systems for extensions of propositional Gödel logic with quantifiers of various types and modalities (see [6] for a survey). In particular, an extension of **HG** for the standard first-order Gödel logic (called **HIF**) was introduced in [8]. Following the work that started in [2], the framework of hypersequents was used by Metcalfe, Ciabattoni, and others for other fuzzy logics (like Łukasiewicz infinite-valued logic), and nowadays it is the major framework for the proof theory of fuzzy logics (see [17]).

Until recently, in all the works about **HG** and its extensions the proofs of completeness (either for the Gödel's many-valued semantics or for the Kripke semantics) and the proofs of cut-elimination have completely been separated. Completeness has been shown for the full calculus (including cut), while cut-elimination has been proved syntactically by some type of induction on complexity of proofs.[2] On the contrary, the recent [4] provided for the first time a constructive, direct, and simple proof of the completeness of the cut-free part of **HG** for its intended semantics (thereby proving both completeness of the calculus and the admissibility of the cut rule in it)[3]. However, [4] did not deal with the first-order extension of **HG**, and it was not clear how to adapt its completeness proof to the first-order case.

In this paper we present a hypersequent system for the standard first-order Gödel logic, for which it is possible to provide a purely semantic, simple (and easy to verify) proof of cut-admissibility. As usual, this proof is actually a completeness proof of the cut-free part of our system for its intended semantics. To overcome the difficulties encountered in adapting the proof of [4] to the first-order case, we move to the *multiple-conclusion* framework. The proposed system, which we call **MCG**, is a multiple-conclusion hypersequent system, which can be seen as a combination of **HIF** and the well-known multiple-conclusion sequent system for intuitionistic logic (called **LJ′** in [20]). Our results apply also to derivations from assumptions, as we actually prove *strong* cut-admissibility,

[2] The syntactic methods are notoriously prone to errors, especially (but certainly not only) in the case of hypersequent systems. Thus the first proof (in [8]) of cut-elimination for **HIF** was erroneous. There has also been a gap in the proof given in [2] in its handling of the case of disjunction. Many other examples, also for ordinary sequential calculi, can be given.

[3] A semantic proof of cut-admissibility for **HG** has been given in [9]. However, a complicated algebraic phase semantics was used there, and the proof is not constructive.

proving that derivations can be confined to those in which cuts are made only on formulas which occur in the assumptions. Finally, at the end of the paper we return to the original single-conclusion system **HIF** for Gödel logic, and use our results about **MCG** to provide a new, semantic proof that this system too admits (strong) cut-admissibility.

2 Preliminaries

Let $\mathcal{L}$ be a first-order language. We assume that the set of free variables and the set of bounded variables are disjoint. We use the metavariable a to range over the free variables, x to range over the bounded variables, p to range over the predicate symbols of $\mathcal{L}$, c to range over its constant symbols, and f to range over its function symbols. The sets of $\mathcal{L}$-terms and $\mathcal{L}$-formulas are defined as usual, and are denoted by $trm_{\mathcal{L}}$ and $frm_{\mathcal{L}}$, respectively. $trm^{cl}_{\mathcal{L}}$ and $frm^{cl}_{\mathcal{L}}$ respectively denote the sets of closed $\mathcal{L}$-terms and closed $\mathcal{L}$-formulas. Given an $\mathcal{L}$-formula ψ, a free-variable a, and an $\mathcal{L}$-term t, we denote by $\psi\{t/a\}$ the $\mathcal{L}$-formula obtained from ψ by replacing all occurrences of a by t.

2.1 Proof-Theoretical Preliminaries

Definition 1. A *sequent* is an ordered pair of finite sets of $\mathcal{L}$-formulas. A *hypersequent* is a finite set of sequents.

Given a set $\mathcal{H}$ of hypersequents, we denote by $frm[\mathcal{H}]$ the set of formulas that appear in $\mathcal{H}$. We shall use the usual sequent notation $\Gamma \Rightarrow \Delta$, and the usual hypersequent notation $s_1 \mid \ldots \mid s_n$. We also employ the standard abbreviations, e.g. $\Gamma, \psi \Rightarrow \Delta$ instead of $\Gamma \cup \{\psi\} \Rightarrow \Delta$, and $H \mid s$ instead of $H \cup \{s\}$.

Definition 2. *A sequent $\Gamma \Rightarrow \Delta$ is* single-conclusion *if Δ contains at most one formula. A $\mathcal{L}$-hypersequent $s_1 \mid \ldots \mid s_n$ is* single-conclusion *if $s_1, \ldots, s_n$ are all single-conclusion.*

Next we review the single-conclusion hypersequent system **HIF** for the standard first-order Gödel logic from [8].[4]

$$\varphi \Rightarrow \varphi \qquad \bot \Rightarrow$$

$$(IW\Rightarrow)\ \frac{H \mid \Gamma \Rightarrow E}{H \mid \Gamma, \psi \Rightarrow E} \qquad (\Rightarrow IW)\ \frac{H \mid \Gamma \Rightarrow}{H \mid \Gamma \Rightarrow \psi} \qquad (EW)\ \frac{H}{H \mid \Gamma \Rightarrow E}$$

$$(com)\ \frac{H_1 \mid \Gamma_1, \Gamma_1' \Rightarrow E_1 \qquad H_2 \mid \Gamma_2, \Gamma_2' \Rightarrow E_2}{H_1 \mid H_2 \mid \Gamma_1, \Gamma_2' \Rightarrow E_1 \mid \Gamma_2, \Gamma_1' \Rightarrow E_2}$$

[4] What we present is actually an equivalent version of the system presented in [8]. Thus $\neg\varphi$ is defined here as $\varphi \supset \bot$, while the density rule is not present, since it can be eliminated. Other insignificant differences are due to the facts that we define hypersequents as *sets* of sequents rather than as *multisets*, and that we use multiplicative versions of the rules rather than additive ones.

$$(cut)\ \frac{H_1 \mid \Gamma_1 \Rightarrow \varphi \quad H_2 \mid \Gamma_2, \varphi \Rightarrow E}{H_1 \mid H_2 \mid \Gamma_1, \Gamma_2 \Rightarrow E}$$

$$(\supset\Rightarrow)\ \frac{H_1 \mid \Gamma_1 \Rightarrow \psi_1 \quad H_2 \mid \Gamma_2, \psi_2 \Rightarrow E}{H_1 \mid H_2 \mid \Gamma_1, \Gamma_2, \psi_1 \supset \psi_2 \Rightarrow E} \qquad (\Rightarrow\supset)\ \frac{H \mid \Gamma, \psi_1 \Rightarrow \psi_2}{H \mid \Gamma \Rightarrow \psi_1 \supset \psi_2}$$

$$(\vee\Rightarrow)\ \frac{H_1 \mid \Gamma_1, \psi_1 \Rightarrow E_1 \quad H_2 \mid \Gamma_2, \psi_2 \Rightarrow E_2}{H_1 \mid H_2 \mid \Gamma_1, \Gamma_2, \psi_1 \vee \psi_2 \Rightarrow E_1, E_2}$$

$$(\Rightarrow\vee_1)\ \frac{H \mid \Gamma \Rightarrow \psi_1}{H \mid \Gamma \Rightarrow \psi_1 \vee \psi_2} \qquad (\Rightarrow\vee_2)\ \frac{H \mid \Gamma \Rightarrow \psi_2}{H \mid \Gamma \Rightarrow \psi_1 \vee \psi_2}$$

$$(\wedge\Rightarrow_1)\ \frac{H \mid \Gamma, \psi_1 \Rightarrow E}{H \mid \Gamma, \psi_1 \wedge \psi_2 \Rightarrow E} \qquad (\wedge\Rightarrow_2)\ \frac{H \mid \Gamma, \psi_2 \Rightarrow E}{H \mid \Gamma, \psi_1 \wedge \psi_2 \Rightarrow E}$$

$$(\Rightarrow\wedge)\ \frac{H_1 \mid \Gamma_1 \Rightarrow \psi_1 \quad H_2 \mid \Gamma_2 \Rightarrow \psi_2}{H_1 \mid H_2 \mid \Gamma_1, \Gamma_2 \Rightarrow \psi_1 \wedge \psi_2}$$

$$(\forall\Rightarrow)\ \frac{H \mid \Gamma, \varphi\{t/a\} \Rightarrow E}{H \mid \Gamma, \forall x(\varphi\{x/a\}) \Rightarrow E} \qquad (\Rightarrow\forall)\ \frac{H \mid \Gamma \Rightarrow \varphi}{H \mid \Gamma \Rightarrow \forall x(\varphi\{x/a\})}$$

$$(\exists\Rightarrow)\ \frac{H \mid \Gamma, \varphi \Rightarrow E}{H \mid \Gamma, \exists x(\varphi\{x/a\}) \Rightarrow E} \qquad (\Rightarrow\exists)\ \frac{H \mid \Gamma \Rightarrow \varphi\{t/a\}}{H \mid \Gamma \Rightarrow \exists x(\varphi\{x/a\})}$$

The rules $(\Rightarrow\forall)$ and $(\exists\Rightarrow)$ must obey the eigenvariable condition: a must not occur in the lower hypersequent. E, E_1 and E_2 denote here sets of formulas containing at most one formula. Note that the sets of formulas denoted by $\Gamma_1, \Gamma_2, \Gamma_1', \Gamma_2'$ need not to be disjoint. Also note that in $(\vee\Rightarrow)$: either $E_1 = E_2$ or one of them is empty.

2.2 Semantic Preliminaries

In this paper we use the usual Kripke-style semantics for the standard first-order Gödel logic, rather than the many-valued one. There are two differences between this semantics and the Kripke-style semantics of first-order intuitionistic logic. First, for Gödel logic we use *linearly* ordered Kripke frames. Second, for first-order Gödel logic we need to use a constant domain, i.e. the same domain in each world, rather than the expanding domains used for intuitionistic logic.[5]

Definition 3. An $\mathcal{L}$-*structure* M is a pair $\langle D, I\rangle$ where D is a nonempty domain and I is an interpretation of constants and function symbols of $\mathcal{L}$, such that $I(c) \in D$ for every constant symbol c of $\mathcal{L}$, and $I(f) \in D^n \to D$ for every n-ary function symbol f of $\mathcal{L}$.

Definition 4. A $\langle \mathcal{L}, D\rangle$-predicate interpretation is a function assigning a subset of D^n to every n-ary predicate symbol of $\mathcal{L}$.

Definition 5. An $\mathcal{L}$-*frame* is a tuple $\mathcal{W} = \langle W, \leq, M, \{I_w\}_{w\in W}\rangle$ where:

1. W is a nonempty set linearly ordered by $\leq$.
2. $M = \langle D, I\rangle$ is an $\mathcal{L}$-structure.

[5] Currently no cut-free hypersequent calculus is known for the logic of linearly ordered Kripke frames with (non-constant) expanding domains.

3. For every $w \in W$, I_w is an $\langle \mathcal{L}, D\rangle$-predicate interpretation.
4. $I_u(p) \subseteq I_w(p)$ for every elements u, w of W such that $u \leq w$, and for every predicate symbol p.

Definition 6. An $\langle \mathcal{L}, D\rangle$-evaluation is a function assigning an element in D to every free variable of $\mathcal{L}$. Given an $\langle \mathcal{L}, D\rangle$-evaluation e, a free variable a, and $d \in D$, we denote by $e_{[a:=d]}$ the $\langle \mathcal{L}, D\rangle$-evaluation which is identical to e except that $e_{[a:=d]}(a) = d$.

Given a structure $M = \langle D, I\rangle$, the M-extension of an $\langle \mathcal{L}, D\rangle$-evaluation e is a function $e' : \mathit{trm}_{\mathcal{L}} \to D$ defined as follows: $e'(c) = I(c)$ for every constant symbol c; $e'(a) = e(a)$ for every free variable a; and $e(f(t_1, \ldots, t_n) = I(f)(e'(t_1), \ldots, e'(t_n))$ for every function symbol f and $t_1, \ldots, t_n \in \mathit{trm}_{\mathcal{L}}$.

Definition 7. Let $\mathcal{W} = \langle W, \leq, M = \langle D, I\rangle, \{I_w\}_{w\in W}\rangle$ be an $\mathcal{L}$-frame, and e be an $\langle \mathcal{L}, D\rangle$-evaluation. The satisfaction relation $\vDash$ is recursively defined as follows:

1. $\mathcal{W}, w, e \vDash p(t_1, \ldots, t_n)$ iff $\langle e'(t_1), \ldots, e'(t_n)\rangle \in I_w(p)$, where e' is the M-extension of e.
2. $\mathcal{W}, w, e \nvDash \bot$.
3. $\mathcal{W}, w, e \vDash \psi_1 \supset \psi_2$ iff $\mathcal{W}, u, e \nvDash \psi_1$ or $\mathcal{W}, u, e \vDash \psi_2$ for every element $u \geq w$.
4. $\mathcal{W}, w, e \vDash \psi_1 \vee \psi_2$ iff $\mathcal{W}, w, e \vDash \psi_1$ or $\mathcal{W}, w, e \vDash \psi_2$.
5. $\mathcal{W}, w, e \vDash \psi_1 \wedge \psi_2$ iff $\mathcal{W}, w, e \vDash \psi_1$ and $\mathcal{W}, w, e \vDash \psi_2$.
6. $\mathcal{W}, w, e \vDash \forall x(\psi\{x/a\})$ iff $\mathcal{W}, w, e_{[a:=d]} \vDash \psi$ for every $d \in D$.
7. $\mathcal{W}, w, e \vDash \exists x(\psi\{x/a\})$ iff $\mathcal{W}, w, e_{[a:=d]} \vDash \psi$ for some $d \in D$.

$\vDash$ is extended to sequents as follows: $\mathcal{W}, w, e \vDash \Gamma \Rightarrow \Delta$ iff either $\mathcal{W}, w, e \nvDash \varphi$ for some $\varphi \in \Gamma$, or $\mathcal{W}, w, e \vDash \varphi$ for some $\varphi \in \Delta$.

It is a routine matter to prove the following proposition:

Proposition 1. *Let $\mathcal{W} = \langle W, \leq, M = \langle D, I\rangle, \{I_w\}_{w\in W}\rangle$ be an $\mathcal{L}$-frame, and e be an $\langle \mathcal{L}, D\rangle$-evaluation. Let ψ be an $\mathcal{L}$-formula, and u be an element of W such that $\mathcal{W}, u, e \vDash \psi$. Then, $\mathcal{W}, w, e \vDash \psi$ for every element w of W such that $u \leq w$.*

Definition 8. Let $\mathcal{W} = \langle W, \leq, M = \langle D, I\rangle, \{I_w\}_{w\in W}\rangle$ be an $\mathcal{L}$-frame.

1. $\mathcal{W}$ is a *model* of a hypersequent H iff for every $\langle \mathcal{L}, D\rangle$-evaluation e, there exists a component $s \in H$ such that $\mathcal{W}, w, e \vDash s$ for every $w \in W$.
2. $\mathcal{W}$ is a model of a set of hypersequents $\mathcal{H}$ iff it is a model of every $H \in \mathcal{H}$.

We define the semantic consequence relation between hypersequents:

Definition 9. Let $\mathcal{H} \cup \{H\}$ be a set of hypersequents. $\mathcal{H} \vdash^{Kr} H$ iff every $\mathcal{L}$-frame which is a model of $\mathcal{H}$ is also a model of H.

3 The Multiple-Conclusion System

The system **MCG** is the following (multiple-conclusion) hypersequent system:

Axioms:

$$\varphi \Rightarrow \varphi \qquad \bot \Rightarrow$$

Structural Rules:

$$(IW\Rightarrow)\ \frac{H \mid \Gamma \Rightarrow \Delta}{H \mid \Gamma, \psi \Rightarrow \Delta} \quad (\Rightarrow IW)\ \frac{H \mid \Gamma \Rightarrow \Delta}{H \mid \Gamma \Rightarrow \Delta, \psi} \quad (EW)\ \frac{H}{H \mid \Gamma \Rightarrow \Delta}$$

$$(com)\ \frac{H_1 \mid \Gamma_1, \Gamma_1' \Rightarrow \Delta_1 \quad H_2 \mid \Gamma_2, \Gamma_2' \Rightarrow \Delta_2}{H_1 \mid H_2 \mid \Gamma_1, \Gamma_2' \Rightarrow \Delta_1 \mid \Gamma_2, \Gamma_1' \Rightarrow \Delta_2} \qquad (split)\ \frac{H \mid \Gamma \Rightarrow \Delta_1, \Delta_2}{H \mid \Gamma \Rightarrow \Delta_1 \mid \Gamma \Rightarrow \Delta_2}$$

$$(cut)\ \frac{H_1 \mid \Gamma_1 \Rightarrow \Delta_1, \varphi \quad H_2 \mid \Gamma_2, \varphi \Rightarrow \Delta_2}{H_1 \mid H_2 \mid \Gamma_1, \Gamma_2 \Rightarrow \Delta_1, \Delta_2}$$

Logical Rules:

$$(\supset\Rightarrow)\ \frac{H_1 \mid \Gamma_1 \Rightarrow \Delta_1, \psi_1 \quad H_2 \mid \Gamma_2, \psi_2 \Rightarrow \Delta_2}{H_1 \mid H_2 \mid \Gamma_1, \Gamma_2, \psi_1 \supset \psi_2 \Rightarrow \Delta_1, \Delta_2} \qquad (\Rightarrow\supset)\ \frac{H \mid \Gamma, \psi_1 \Rightarrow \psi_2}{H \mid \Gamma \Rightarrow \psi_1 \supset \psi_2}$$

$$(\vee\Rightarrow)\ \frac{H_1 \mid \Gamma_1, \psi_1 \Rightarrow \Delta_1 \quad H_2 \mid \Gamma_2, \psi_2 \Rightarrow \Delta_2}{H_1 \mid H_2 \mid \Gamma_1, \Gamma_2, \psi_1 \vee \psi_2 \Rightarrow \Delta_1, \Delta_2} \qquad (\Rightarrow\vee)\ \frac{H \mid \Gamma \Rightarrow \Delta, \psi_1, \psi_2}{H \mid \Gamma \Rightarrow \Delta, \psi_1 \vee \psi_2}$$

$$(\wedge\Rightarrow)\ \frac{H \mid \Gamma, \psi_1, \psi_2 \Rightarrow \Delta}{H \mid \Gamma, \psi_1 \wedge \psi_2 \Rightarrow \Delta} \qquad (\Rightarrow\wedge)\ \frac{H_1 \mid \Gamma_1 \Rightarrow \Delta_1, \psi_1 \quad H_2 \mid \Gamma_2 \Rightarrow \Delta_2, \psi_2}{H_1 \mid H_2 \mid \Gamma_1, \Gamma_2 \Rightarrow \Delta_1, \Delta_2, \psi_1 \wedge \psi_2}$$

$$(\forall\Rightarrow)\ \frac{H \mid \Gamma, \varphi\{t/a\} \Rightarrow \Delta}{H \mid \Gamma, \forall x(\varphi\{x/a\}) \Rightarrow \Delta} \qquad (\Rightarrow\forall)\ \frac{H \mid \Gamma \Rightarrow \Delta, \varphi}{H \mid \Gamma \Rightarrow \Delta, \forall x(\varphi\{x/a\})}$$

$$(\exists\Rightarrow)\ \frac{H \mid \Gamma, \varphi \Rightarrow \Delta}{H \mid \Gamma, \exists x(\varphi\{x/a\}) \Rightarrow \Delta} \qquad (\Rightarrow\exists)\ \frac{H \mid \Gamma \Rightarrow \Delta, \varphi\{t/a\}}{H \mid \Gamma \Rightarrow \Delta, \exists x(\varphi\{x/a\})}$$

The rules $(\Rightarrow\forall)$ and $(\exists\Rightarrow)$ must obey the eigenvariable condition: a must not occur in the lower hypersequent.

Remark 1. The main difference between **MCG** and the system **HIF** of [8] is the fact that **MCG** employs *multiple-conclusion* hypersequents. Among other things, this involves having full internal weakening $(\Rightarrow IW)$ on the right, and allowing also right context formulas in all the rules, except for $(\Rightarrow\supset)$. Note that such formulas are *not* allowed in $(\Rightarrow\supset)$, and so this rule looks exactly like its single-conclusion counterpart. The communication rule is also strengthened, allowing arbitrary finite sets of formula in the right-hand side of its premises. In addition, an extra (right) split rule is added, allowing to split the formulas in the right side of one component into two different components.

Definition 10. Let $\mathcal{H} \cup \{H\}$ be a set of hypersequents. $\mathcal{H} \vdash H$ if there exists a derivation of H from $\mathcal{H}$ in **MCG**. Given a set $\mathcal{E}$ of $\mathcal{L}$-formulas, we write $\mathcal{H} \vdash^{\mathcal{E}} H$ if there exists a derivation of H from $\mathcal{H}$ in **MCG** in which the cut-formula of every application of the rule is in $\mathcal{E}$.

Remark 2. In this notation, strong cut-admissibility means that $\mathcal{H} \vdash^{frm[\mathcal{H}]} H$ whenever $\mathcal{H} \vdash H$. Usual cut-admissibility is obtained as a special case when $\mathcal{H} = \emptyset$.

The following usual lemma will be used in the sequel.

Lemma 1. *Let $\mathcal{H} \cup \{H\}$ be a set of hypersequents, c be a constant symbol not occurring in $\mathcal{H} \cup \{H\}$, and a be a free variable. Then, $\mathcal{H} \vdash^{frm[\mathcal{H}]} H$ iff $\mathcal{H}\{c/a\} \vdash^{frm[\mathcal{H}]} H\{c/a\}$.*

4 Soundness, Completeness and Cut-Admissibility

In this section we prove the soundness and completeness theorem for **MCG** with respect to the Kripke semantics of first-order Gödel logic (presented in Section 2). Completeness is proved for **MCG** without the cut-rule, and so it also proves cut-admissibility. We begin with the soundness theorem.

Theorem 1. *Let $\mathcal{H} \cup \{H\}$ be a set of closed hypersequent. If $\mathcal{H} \vdash^{frm[\mathcal{H}]} H$ then $\mathcal{H} \vdash^{Kr} H$.*

Proof (Outline). Let $\mathcal{W} = \langle W, \leq, M = \langle D, I\rangle, \{I_w\}_{w \in W}\rangle$ be an $\mathcal{L}$-frame which is a model of $\mathcal{H}$. We show that for every $\langle \mathcal{L}, D\rangle$-evaluation e, there exists a component $s \in H$ such that $\mathcal{W}, w, e \vDash s$ for every $w \in W$. Since the axioms of **MCG** and the assumptions of $\mathcal{H}$ trivially have this property, it suffices to show that this property is preserved also by applications of the rules of **MCG**. This is a routine matter. We do here only the case of (com).

Suppose that $H = H_1 \mid H_2 \mid \Gamma_1, \Gamma_2' \Rightarrow \Delta_1 \mid \Gamma_2, \Gamma_1' \Rightarrow \Delta_2$ is derived from the hypersequents $H_1 \mid \Gamma_1, \Gamma_1' \Rightarrow \Delta_1$ and $H_2 \mid \Gamma_2, \Gamma_2' \Rightarrow \Delta_2$ using (com). Assume for contradiction that $\mathcal{W}$ is not a model of H. Thus there exists an $\langle \mathcal{L}, D\rangle$-evaluation e, such that for every $s \in H$, there exists $w \in W$ such that $\mathcal{W}, w, e \nvDash s$. In particular, for every $s \in H_1 \cup H_2$, there exists $w \in W$ such that $\mathcal{W}, w, e \nvDash s$. In addition, there exist $w_1 \in W$ such that $\mathcal{W}, w_1, e \nvDash \Gamma_1, \Gamma_2' \Rightarrow \Delta_1$, and $w_2 \in W$ such that $\mathcal{W}, w_2, e \nvDash \Gamma_2, \Gamma_1' \Rightarrow \Delta_2$. By definition, $\mathcal{W}, w_1, e \vDash \psi$ for every $\psi \in \Gamma_1 \cup \Gamma_2'$, $\mathcal{W}, w_1, e \nvDash \psi$ for every $\psi \in \Delta_1$, $\mathcal{W}, w_2, e \vDash \psi$ for every $\psi \in \Gamma_2 \cup \Gamma_1'$, and $\mathcal{W}, w_2, e \nvDash \psi$ for every $\psi \in \Delta_2$. Since $\leq$ is linear, either $w_1 \leq w_2$ or $w_2 \leq w_1$. Assume w.l.o.g that $w_1 \leq w_2$. Then by Proposition 1, $\mathcal{W}, w_2, e \vDash \psi$ for every $\psi \in \Gamma_2'$. It follows that $\mathcal{W}, w_2, e \nvDash \Gamma_2, \Gamma_2' \Rightarrow \Delta_2$. But this implies that $\mathcal{W}$ is not a model of $H_2 \mid \Gamma_2, \Gamma_2' \Rightarrow \Delta_2$. □

To prove completeness, we use *extended sequents* and *extended hypersequents*, defined as follows:

Definition 11. An *extended sequent* is an ordered pair of (possibly infinite) sets of $\mathcal{L}$-formulas, written: $\mathcal{T} \Rightarrow \mathcal{U}$. Given two extended sequents $\mu_1 = \mathcal{T}_1 \Rightarrow \mathcal{U}_1$ and $\mu_2 = \mathcal{T}_2 \Rightarrow \mathcal{U}_2$, we write $\mu_1 \sqsubseteq \mu_2$ if $\mathcal{T}_1 \subseteq \mathcal{T}_2$ and $\mathcal{U}_1 \subseteq \mathcal{U}_2$.

Definition 12. An *extended hypersequent* is a (possibly infinite) set of extended sequents. Given two extended hypersequents Ω_1, Ω_2, we write $\Omega_1 \sqsubseteq \Omega_2$ (and say that Ω_2 *extends* Ω_1) if for every extended sequent $\mu_1 \in \Omega_1$, there exists $\mu_2 \in \Omega_2$ such that $\mu_1 \sqsubseteq \mu_2$.

We shall use the same notations for extended sequents and extended hypersequents. For example, we write $\Omega \mid s$ instead of $\Omega \cup \{s\}$.

Definition 13. An extended sequent $\mathcal{T} \Rightarrow \mathcal{U}$ admits *the witness property* if the following hold:

1. If $\forall x(\psi\{x/a\}) \in \mathcal{U}$ then there exists a constant c such that $\psi\{c/a\} \in \mathcal{U}$.
2. If $\exists x(\psi\{x/a\}) \in \mathcal{T}$ then there exists a constant c such that $\psi\{c/a\} \in \mathcal{T}$.

Definition 14. Let Ω be an extended hypersequent, and $\mathcal{H}$ be a set of hypersequents.

1. Ω is called *closed* if it consists of extended sequents consisting only of closed $\mathcal{L}$-formulas.
2. Ω is called *$\mathcal{H}$-consistent* if $\mathcal{H} \nvdash^{frm[\mathcal{H}]} H$ for every hypersequent $H \sqsubseteq \Omega$.
3. Let ψ be an $\mathcal{L}$-formula. Ω is called *internally $\mathcal{H}$-maximal with respect to ψ* if for every $\mathcal{T} \Rightarrow \mathcal{U} \in \Omega$:
   (a) If $\psi \notin \mathcal{T}$ then $\Omega \mid \mathcal{T}, \psi \Rightarrow \mathcal{U}$ is not $\mathcal{H}$-consistent.
   (b) If $\psi \notin \mathcal{U}$ then $\Omega \mid \mathcal{T} \Rightarrow \mathcal{U}, \psi$ is not $\mathcal{H}$-consistent.
4. Ω is called *internally $\mathcal{H}$-maximal* if it is internally $\mathcal{H}$-maximal with respect to any closed $\mathcal{L}$-formula.
5. Let s be a sequent of the form $\psi_1 \Rightarrow \psi_2$. Ω is called *externally $\mathcal{H}$-maximal with respect to s* if either $\{s\} \sqsubseteq \Omega$, or $\Omega \mid s$ is not $\mathcal{H}$-consistent.
6. Ω is called *externally $\mathcal{H}$-maximal* if it is externally $\mathcal{H}$-maximal with respect to any closed sequent of the form $\psi_1 \Rightarrow \psi_2$.
7. Ω admits *the witness property* if every $\mu \in \Omega$ admits the witness property.
8. Ω is called *$\mathcal{H}$-maximal* if it is closed, $\mathcal{H}$-consistent, internally $\mathcal{H}$-maximal, externally $\mathcal{H}$-maximal, and it admits the witness property.

Obviously, every hypersequent is an extended hypersequent, and so all of these properties apply to (usual) hypersequents as well.

The following three propositions are easily proved in the presence of the internal and external weakening rules:

Proposition 2. *A usual hypersequent H is $\mathcal{H}$-consistent iff $\mathcal{H} \nvdash^{frm[\mathcal{H}]} H$.*

Proposition 3. *Let Ω be an extended hypersequent, which is internally $\mathcal{H}$-maximal with respect to a formula ψ. For every $\mathcal{T} \Rightarrow \mathcal{U} \in \Omega$:*

1. *If $\psi \notin \mathcal{T}$, then $\mathcal{H} \vdash^{frm[\mathcal{H}]} H \mid \Gamma, \psi \Rightarrow \Delta$ for some hypersequent $H \sqsubseteq \Omega$ and sequent $\Gamma \Rightarrow \Delta \sqsubseteq \mathcal{T} \Rightarrow \mathcal{U}$.*
2. *If $\psi \notin \mathcal{U}$, then $\mathcal{H} \vdash^{frm[\mathcal{H}]} H \mid \Gamma \Rightarrow \Delta, \psi$ for some hypersequent $H \sqsubseteq \Omega$ and sequent $\Gamma \Rightarrow \Delta \sqsubseteq \mathcal{T} \Rightarrow \mathcal{U}$.*

Proposition 4. *Let Ω be an extended hypersequent, which is externally $\mathcal{H}$-maximal with respect to a sequent s. Then, either $s \sqsubseteq \Omega$, or there exists a hypersequent $H \sqsubseteq \Omega$ such that $\mathcal{H} \vdash^{frm[\mathcal{H}]} H \mid s$.*

A certain $\mathcal{H}$-maximal extended hypersequent serves as the set of worlds in the refuting frame built in the completeness proof. Lemma 4 below ensures the existence of that extended hypersequent. In turn, for the proof of Lemma 4 we need Lemmas 2 and 3 below.

Lemma 2. *Assume $\mathcal{L}$ has an infinite number of constant symbols. Let $\mathcal{H}$ be a set of hypersequents, and $H = \Gamma_1 \Rightarrow \Delta_1 \mid \ldots \mid \Gamma_n \Rightarrow \Delta_n$ be a $\mathcal{H}$-consistent closed hypersequent. Then there exists a $\mathcal{H}$-consistent closed hypersequent H' of the form $\Gamma'_1 \Rightarrow \Delta'_1 \mid \ldots \mid \Gamma'_n \Rightarrow \Delta'_n$, such that $\Gamma_i \subseteq \Gamma'_i$ and $\Delta_i \subseteq \Delta'_i$ for every $1 \leq i \leq n$, and H' admits the witness property.*

Lemma 3. *Assume $\mathcal{L}$ has an infinite number of constant symbols. Let $\mathcal{H}$ be a set of hypersequents, and $H = \Gamma_1 \Rightarrow \Delta_1 \mid \ldots \mid \Gamma_n \Rightarrow \Delta_n$ be a $\mathcal{H}$-consistent closed hypersequent. Let ψ be a closed $\mathcal{L}$-formula, and s be a closed sequent of the form $\psi_1 \Rightarrow \psi_2$. Then there exists a $\mathcal{H}$-consistent closed hypersequent H', such that:*

- *$H' = \Gamma'_1 \Rightarrow \Delta'_1 \mid \ldots \mid \Gamma'_{n'} \Rightarrow \Delta'_{n'}$, where $n' \in \{n, n+1\}$, $\Gamma_i \subseteq \Gamma'_i$ and $\Delta_i \subseteq \Delta'_i$ for every $1 \leq i \leq n$.*
- *H' is internally $\mathcal{H}$-maximal with respect to ψ.*
- *H' is externally $\mathcal{H}$-maximal with respect to s.*
- *H' admits the witness property.*

Lemma 4. *Assume $\mathcal{L}$ has an infinite number of constant symbols. Let $\mathcal{H}$ be a set of hypersequents. Every $\mathcal{H}$-consistent closed hypersequent H can be extended to a $\mathcal{H}$-maximal extended hypersequent Ω.*

Using Lemma 4, we turn to the completeness of the cut-free fragment of **MCG**.

Theorem 2. *Let $\mathcal{H}_0 \cup \{H_0\}$ be a set of closed hypersequent. If $\mathcal{H}_0 \vdash^{Kr} H_0$ then $\mathcal{H}_0 \vdash^{frm[\mathcal{H}]} H_0$.*

Proof. Assume $\mathcal{H}_0 \not\vdash^{\mathcal{E}} H_0$, where $\mathcal{E} = frm[\mathcal{H}_0]$. We construct an $\mathcal{L}$-frame $\mathcal{W}$ which is a model of $\mathcal{H}_0$ but not of H_0. First, assume (w.l.o.g) that $\mathcal{L}$ has an infinite number of constant symbols (if not, then we add infinitely many constant symbols, and obviously $\mathcal{H}_0 \not\vdash^{\mathcal{E}} H_0$ still holds). By Lemma 4, there exists a $\mathcal{H}_0$-maximal extended hypersequent Ω such that $H_0 \sqsubseteq \Omega$.

Define $\mathcal{W} = \langle W, \leq, M, \{I_w\}_{w \in W} \rangle$, as follows:

- $W = \Omega$ (obviously, W is not empty).
- For every $\mathcal{T}_1 \Rightarrow \mathcal{U}_1, \mathcal{T}_2 \Rightarrow \mathcal{U}_2 \in W$, $\mathcal{T}_1 \Rightarrow \mathcal{U}_1 \leq \mathcal{T}_2 \Rightarrow \mathcal{U}_2$ iff $\mathcal{T}_1 \subseteq \mathcal{T}_2$.
- $M = \langle D, I \rangle$ where D is the set of all closed $\mathcal{L}$-terms, $I(c) = c$ for every constant c, and $I(f)(t_1, \ldots, t_n) = f(t_1, \ldots, t_n)$ for every n-ary function symbol f and $t_1, \ldots, t_n \in D$.
- $\langle t_1, \ldots, t_n \rangle \in I_{\mathcal{T} \Rightarrow \mathcal{U}}(p)$ iff $p(t_1, \ldots, t_n) \in \mathcal{T}$ for every n-ary predicate symbol p and $t_1, \ldots, t_n \in D$.

We first prove that $\langle W, \leq \rangle$ is linearly ordered:

Partial Order. Obviously $\leq$ is reflexive and transitive. To see that it is also anti-symmetric, let $w_1, w_2 \in W$ such that $w_1 \leq w_2$ and $w_2 \leq w_1$. Assume $w_1 = \mathcal{T}_1 \Rightarrow \mathcal{U}_1$ and $w_2 = \mathcal{T}_2 \Rightarrow \mathcal{U}_2$. By definition, $\mathcal{T}_1 = \mathcal{T}_2$ in this case. Assume for contradiction that $\mathcal{U}_1 \neq \mathcal{U}_2$, and let $\psi \in \mathcal{U}_1 \setminus \mathcal{U}_2$ (w.l.o.g.). Since Ω is internally $\mathcal{H}$-maximal, there exist a hypersequent $H \sqsubseteq \Omega$ and a sequent $\Gamma \Rightarrow \Delta \sqsubseteq w_2$, such that $\mathcal{H}_0 \vdash^{\mathcal{E}} H \mid \Gamma \Rightarrow \Delta, \psi$. Using the split rule, we obtain $\mathcal{H}_0 \vdash^{\mathcal{E}} H \mid \Gamma \Rightarrow \psi \mid \Gamma \Rightarrow \Delta$. But, $\Gamma \Rightarrow \psi \sqsubseteq w_1$, and this contradicts Ω's consistency. Hence $\mathcal{U}_1 = \mathcal{U}_2$, and so $w_1 = w_2$.

Linearity. Let $\mathcal{T}_1 \Rightarrow \mathcal{U}_1, \mathcal{T}_2 \Rightarrow \mathcal{U}_2 \in W$. Assume for contradiction that $\mathcal{T}_1 \not\subseteq \mathcal{T}_2$ and $\mathcal{T}_2 \not\subseteq \mathcal{T}_1$. Let $\psi_1 \in \mathcal{T}_1 \setminus \mathcal{T}_2$ and $\psi_2 \in \mathcal{T}_2 \setminus \mathcal{T}_1$. By Ω's internal maximality, there exist hypersequents $H_1, H_2 \sqsubseteq \Omega$ and sequents $\Gamma_1 \Rightarrow \Delta_1 \sqsubseteq \mathcal{T}_1 \Rightarrow \mathcal{U}_1$ and $\Gamma_2 \Rightarrow \Delta_2 \sqsubseteq \mathcal{T}_2 \Rightarrow \mathcal{U}_2$ such that $\mathcal{H}_0 \vdash^{\mathcal{E}} H_1 \mid \Gamma_1, \psi_2 \Rightarrow \Delta_1$ and such that $\mathcal{H}_0 \vdash^{\mathcal{E}} H_2 \mid \Gamma_2, \psi_1 \Rightarrow \Delta_2$. By applying (com) to these two hypersequents we obtain $\mathcal{H}_0 \vdash^{\mathcal{E}} H_1 \mid H_2 \mid \Gamma_1, \psi_1 \Rightarrow \Delta_1 \mid \Gamma_2, \psi_2 \Rightarrow \Delta_2$. But this contradicts Ω's consistency.

The following claims are proved by a standard structural induction:

- For every $\langle \mathcal{L}, D \rangle$-evaluation e and $t \in D$, $e'(t) = t$.
- For every $\langle \mathcal{L}, D \rangle$-evaluation e, $t \in D$, an $\mathcal{L}$-formula ψ, a free variable a, and $w \in W$: $\mathcal{W}, w, e_{[a:=t]} \vDash \psi$ iff $\mathcal{W}, w, e \vDash \psi\{t/a\}$.

Next we prove that the following hold for every $w = \mathcal{T} \Rightarrow \mathcal{U} \in W$, and $\langle \mathcal{L}, D \rangle$-evaluation e:

(a) If $\theta \in \mathcal{T}$ then $\mathcal{W}, w, e \vDash \theta$.
(b) If $\theta \in \mathcal{U}$ then $\mathcal{W}, w, e \nvDash \theta$.

(a) and **(b)** are proved together using a simultaneous induction on the complexity of θ. Here we do three crucial cases.

Let $w = \mathcal{T} \Rightarrow \mathcal{U} \in W$ and let e be an $\langle \mathcal{L}, D \rangle$-evaluation.

- Suppose θ is a closed atomic formula $p(t_1, \ldots, t_n)$. By definition, $\mathcal{W}, w, e \vDash \theta$ iff $\langle e'(t_1), \ldots, e'(t_n) \rangle \in I_w(p)$, where e' is the M-extension of e (see Definition 6). By a previous claim, $e'(t_i) = t_i$ for every $1 \leq i \leq n$. And so, our construction ensures that $\mathcal{W}, w, e \vDash \theta$ iff $\theta \in \mathcal{T}$. This proves (**a**). For (**b**), note that $\psi \Rightarrow \psi$ is an axiom (for every $\mathcal{L}$-formula ψ), and since Ω is $\mathcal{H}$-consistent, $\theta \in \mathcal{U}$ implies $\theta \notin \mathcal{T}$.
- Suppose $\theta = \psi_1 \supset \psi_2$.
  1. Assume that $\theta \in \mathcal{T}$. We show that for every element $w' \in W$ such that $w \leq w'$ either $\mathcal{W}, w', e \nvDash \psi_1$ or $\mathcal{W}, w', e \vDash \psi_2$.
     Let $w' = \mathcal{T}' \Rightarrow \mathcal{U}' \in W$ such that $w \leq w'$ (and so, $\mathcal{T} \subseteq \mathcal{T}'$). By the induction hypothesis, it suffices to show that either $\psi_1 \in \mathcal{U}'$ or $\psi_2 \in \mathcal{T}'$. Assume otherwise. Then by Ω's internal maximality, there exist hypersequents $H_1, H_2 \sqsubseteq \Omega$, and sequents $\Gamma_1 \Rightarrow \Delta_1, \Gamma_2 \Rightarrow \Delta_2 \sqsubseteq \mathcal{T}' \Rightarrow \mathcal{U}'$ such that $\mathcal{H}_0 \vdash^{\mathcal{E}} H_1 \mid \Gamma_1 \Rightarrow \Delta_1, \psi_1$, and $\mathcal{H}_0 \vdash^{\mathcal{E}} H_2 \mid \Gamma_2, \psi_2 \Rightarrow \Delta_2$. By applying

($\supset\Rightarrow$) we obtain $\mathcal{H}_0 \vdash^{\mathcal{E}} H_1 \mid H_2 \mid \Gamma_1, \Gamma_2, \theta \Rightarrow \Delta_1, \Delta_2$. But since $\theta \in \mathcal{T}$, $\theta \in \mathcal{T}'$ and so $H_1 \mid H_2 \mid \Gamma_1, \Gamma_2, \theta \Rightarrow \Delta_1, \Delta_2 \sqsubseteq \Omega$. This contradicts Ω's consistency.

2. Assume that $\theta \in \mathcal{U}$.
   First we claim that $\mathcal{H}_0 \nvdash^{\mathcal{E}} H \mid \psi_1 \Rightarrow \psi_2$ for every hypersequent $H \sqsubseteq \Omega$. To see this assume for contradiction that there exists a hypersequent $H \sqsubseteq \Omega$, such that $\mathcal{H}_0 \vdash^{\mathcal{E}} H \mid \psi_1 \Rightarrow \psi_2$. By applying ($\Rightarrow\supset$) to this hypersequent we obtain $\mathcal{H}_0 \vdash^{\mathcal{E}} H \mid \Rightarrow \theta$. But this contradicts Ω's consistency.
   Therefore, by Ω's external maximality, $\psi_1 \Rightarrow \psi_2 \sqsubseteq \Omega$. Thus there exists an extended sequent $\mathcal{T}' \Rightarrow \mathcal{U}' \in \Omega$, such that $\psi_1 \in \mathcal{T}'$ and $\psi_2 \in \mathcal{U}'$. By the induction hypothesis, $\mathcal{W}, \mathcal{T}' \Rightarrow \mathcal{U}', e \vDash \psi_1$ and $\mathcal{W}, \mathcal{T}' \Rightarrow \mathcal{U}', e \nvDash \psi_2$. It follows that if $\mathcal{T} \subseteq \mathcal{T}'$, then $\mathcal{W}, w, e \nvDash \theta$ and we are done.
   Assume now that $\mathcal{T} \nsubseteq \mathcal{T}'$. By linearity, $\mathcal{T}' \subseteq \mathcal{T}$, and so $\psi_1 \in \mathcal{T}$. By the induction hypothesis, $\mathcal{W}, w, e \vDash \psi_1$. Now notice that $\psi_2 \in \mathcal{U}$. To see this assume for contradiction that $\psi_2 \notin \mathcal{U}$. Then by Ω's internal maximality, there exist a hypersequent $H \sqsubseteq \Omega$, and a sequent $\Gamma \Rightarrow \Delta \sqsubseteq \mathcal{T} \Rightarrow \mathcal{U}$, such that $\mathcal{H}_0 \vdash^{\mathcal{E}} H \mid \Gamma \Rightarrow \Delta, \psi_2$. By applying the split rule we obtain $\mathcal{H}_0 \vdash^{\mathcal{E}} H \mid \Gamma \Rightarrow \Delta \mid \Gamma \Rightarrow \psi_2$. By applying internal weakening we obtain $\mathcal{H}_0 \vdash^{\mathcal{E}} H \mid \Gamma \Rightarrow \Delta \mid \Gamma, \psi_1 \Rightarrow \psi_2$. Finally, by ($\Rightarrow\supset$) we obtain $\mathcal{H}_0 \vdash^{\mathcal{E}} H \mid \Gamma \Rightarrow \Delta \mid \Gamma \Rightarrow \theta$. But this contradicts Ω's consistency. By the induction hypothesis, $\mathcal{W}, w, e \nvDash \psi_2$. This again implies that $\mathcal{W}, w, e \nvDash \theta$.

– Suppose $\theta = \forall x(\psi\{x/a\})$.
  1. Assume that $\mathcal{W}, w, e \nvDash \theta$. We show that $\theta \notin \mathcal{T}$. By definition, there exists some $t \in D$ such that $\mathcal{W}, w, e_{[a:=t]} \nvDash \psi$. By a previous claim, it follows that $\mathcal{W}, w, e \nvDash \psi\{t/a\}$. By the induction hypothesis, $\psi\{t/a\} \notin \mathcal{T}$. Now, using Ω's internal maximality, there exist a hypersequent $H \sqsubseteq \Omega$ and a sequent $\Gamma \Rightarrow \Delta \sqsubseteq \mathcal{T} \Rightarrow \mathcal{U}$, such that $\mathcal{H}_0 \vdash^{\mathcal{E}} H \mid \Gamma, \psi\{t/a\} \Rightarrow \Delta$. By applying ($\forall\Rightarrow$), we obtain $\mathcal{H}_0 \vdash^{\mathcal{E}} H \mid \Gamma, \theta \Rightarrow \Delta$. Since Ω is $\mathcal{H}$-consistent, $\theta \notin \mathcal{T}$.
  2. Assume that $\theta \in \mathcal{U}$. By Ω's witness property, there exists a constant symbol c such that $\psi\{c/a\} \in \mathcal{U}$. From the induction hypothesis it follows that $\mathcal{W}, w, e \nvDash \psi\{c/a\}$. By a previous claim, it follows that $\mathcal{W}, w, e_{[a:=c]} \nvDash \psi$. Since $c \in D$, by definition, $\mathcal{W}, w, e \nvDash \theta$.

It remains to show that $\mathcal{W}$ is a model of $\mathcal{H}_0$ but not of H_0. First, notice that for every $\psi \in \mathit{frm}[\mathcal{H}]$ and $\mathcal{T} \Rightarrow \mathcal{U} \in \Omega$, either $\psi \in \mathcal{T}$ or $\psi \in \mathcal{U}$. To see this, note that otherwise, by Ω's internal maximality, there exist hypersequents $H_1, H_2 \sqsubseteq \Omega$, and sequents $\Gamma_1 \Rightarrow \Delta_1, \Gamma_2 \Rightarrow \Delta_2 \sqsubseteq \mathcal{T} \Rightarrow \mathcal{U}$, such that $\mathcal{H}_0 \vdash^{\mathcal{E}} H_1 \mid \Gamma_1, \psi \Rightarrow \Delta_1$ and $\mathcal{H}_0 \vdash^{\mathcal{E}} H_2 \mid \Gamma_2 \Rightarrow \Delta_2, \psi$. Now using a (legal) application of the cut rule, we obtain $\mathcal{H}_0 \vdash^{\mathcal{E}} H_1 \mid H_2 \mid \Gamma_1, \Gamma_2 \Rightarrow \Delta_1, \Delta_2$, but this contradicts Ω's consistency.

Now let $H \in \mathcal{H}_0$, and let e be an $\langle \mathcal{L}, D \rangle$-evaluation. Since obviously $\mathcal{H}_0 \vdash^{\mathcal{E}} H$, Lemma 2 implies that $H \not\sqsubseteq \Omega$. Thus there exists a sequent $s \in H$, such that $s \not\sqsubseteq \mu$ for every $\mu \in \Omega$. We prove that $\mathcal{W}, w, e \vDash s$ for every $w \in W$. Let $w \in W$. Assume $w = \mathcal{T} \Rightarrow \mathcal{U}$ and $s = \Gamma \Rightarrow \Delta$. Since $s \not\sqsubseteq w$, there either exists $\psi \in \Gamma$ such that $\psi \notin \mathcal{T}$, or $\psi \in \Delta$ such that $\psi \notin \mathcal{U}$. This implies that there either exists $\psi \in \Gamma$

such that $\psi \in \mathcal{U}$, or $\psi \in \Delta$ such that $\psi \in \mathcal{T}$. By **(a)** and **(b)**, either there exists $\psi \in \Gamma$ such that $\mathcal{W}, w, e \not\models \psi$, or there exists $\psi \in \Delta$ such that $\mathcal{W}, w, e \models \psi$. Therefore, $\mathcal{W}, w, e \models s$.

We end the proof by showing that $\mathcal{W}$ is not a model of H_0. Let e be an arbitrary $\langle \mathcal{L}, D \rangle$-evaluation, and let $\Gamma \Rightarrow \Delta \in H_0$. Since $H_0 \sqsubseteq \Omega$ there exists an extended sequent $w = \mathcal{T} \Rightarrow \mathcal{U} \in \Omega$ such that $\Gamma \Rightarrow \Delta \sqsubseteq w$. By **(a)**, for every $\psi \in \Gamma$, $\mathcal{W}, w, e \models \psi$. By **(b)**, for every $\psi \in \Delta$, $\mathcal{W}, w, e \not\models \psi$. Thus, $\mathcal{W}, w, e \not\models \Gamma \Rightarrow \Delta$. □

Finally, we state the two main corollaries, easily obtained from the two previous theorems.

Corollary 1 (Strong Soundness and Completeness)
MCG *is strongly sound and complete with respect to the Kripke semantics of the standard first-order Gödel logic, i.e.* $\mathcal{H} \vdash H$ *iff* $\mathcal{H} \vdash^{Kr} H$ *for every set* $\mathcal{H} \cup \{H\}$ *of closed hypersequent.*

Corollary 2 (Strong Cut-Admissibility)
MCG *admits strong cut-admissibility, i.e.* $\mathcal{H} \vdash H$ *iff* $\mathcal{H} \vdash^{frm[\mathcal{H}]} H$, *for every set* $\mathcal{H} \cup \{H\}$ *of closed hypersequent.*

Remark 3 In [21] the following density rule was introduced and used to axiomatize standard first-order Gödel logic:

$$\frac{\Gamma \Rightarrow \varphi \vee (\psi \supset p) \vee (p \supset \theta)}{\Gamma \Rightarrow \varphi \vee (\psi \supset \theta)}$$

where p (a metavariable for an atomic formula) does not occur in the conclusion. In [19] this rule was proved to be admissible (using a semantic proof). The (single-conclusion) hypersquential version of this rule has the form (see [6]):

$$\frac{H \mid \Gamma \Rightarrow p \mid \Delta, p \Rightarrow \psi}{H \mid \Gamma, \Delta \Rightarrow \psi}$$

By Corollary 1, this rule is admissible in **MCG**.

5 Cut-Admissibility for HIF

In this section we study the relation between **MCG** and the single-conclusion system **HIF**, and derive a semantic cut-admissibility proof for the system **HIF** itself. Denote by $\vdash_{\leq 1}$ the provability relation (between sets of single-conclusion hypersequents, and single-conclusion hypersequents) induced by **HIF** (see Section 2).

Definition 15. Given a hypersequent H, $H^{\leq 1}$ is the single-conclusion hypersequent $\bigcup_{\Gamma \Rightarrow \Delta \in H} \{\Gamma \Rightarrow E \mid E \subseteq \Delta\}$, where E denotes sets of $\mathcal{L}$-formulas which are either singletons or empty. Let $\mathcal{H}^{\leq 1} = \{H^{\leq 1} \mid H \in \mathcal{H}\}$.

The following theorem provides the relation between **MCG** and **HIF**.

Theorem 3. *For every set of hypersequents* $\mathcal{H} \cup \{H\}$*, and set* $\mathcal{E}$ *of* $\mathcal{L}$*-formulas,* $\mathcal{H} \vdash^{\mathcal{E}} H$ *iff* $\mathcal{H}^{\leq 1} \vdash^{\mathcal{E}}_{\leq 1} H^{\leq 1}$*.*

The proof of this theorem is done as usual by induction on the length of the proof in **MCG**. The most problematic case (dealing with the rule $(\exists\Rightarrow)$) follows from Lemma 30 in [6].

Corollary 3 (Strong Cut-Admissibility for HIF).
HIF *admits strong cut-admissibility, i.e.* $\mathcal{H} \vdash_{\leq 1} H$ *iff* $\mathcal{H} \vdash^{frm[\mathcal{H}]}_{\leq 1} H$*, for every set* $\mathcal{H} \cup \{H\}$ *of closed single-conclusion hypersequents.*

Proof. One direction is trivial. For the converse, assume $\mathcal{H} \vdash_{\leq 1} H$. In this case, obviously, $\mathcal{H} \vdash H$. By Corollary 2, $\mathcal{H} \vdash^{frm[\mathcal{H}]} H$. Theorem 3 implies that $\mathcal{H}^{\leq 1} \vdash^{frm[\mathcal{H}]}_{\leq 1} H^{\leq 1}$. Now, notice that for a single-conclusion hypersequent H, $H^{\leq 1} = H \cup \{\Gamma \Rightarrow \emptyset \mid \Gamma \Rightarrow \varphi \in H\}$, and obviously $H^{\leq 1} \vdash^{\emptyset}_{\leq 1} H$ and $H \vdash^{\emptyset}_{\leq 1} H^{\leq 1}$. It now follows that $\mathcal{H} \vdash^{frm[\mathcal{H}]}_{\leq 1} H$. □

6 Further Research

We believe that a (multiple-conclusion) hypersequent system can also be used to provide a similar semantic proof of strong cut-admissibility in Gentzen's *LJ*. Many other (multiple or single-conclusion) hypersequent systems for various propositional and first-order fuzzy logics and intermediate logics have only syntactic proofs of (usual) cut-elimination theorem (see e.g. [17]). It should be interesting to find for them too more simple semantic proofs and derive corresponding strong cut-admissibility theorems. For other fuzzy logics, Kripke-style semantics might not suffice.

Acknowledgements

We are grateful to three anonymous referees for their helpful suggestions and comments. This research was supported by The Israel Science Foundation (grant no. 280-10).

References

1. Avellone, A., Ferrari, M., Miglioli, P.: Duplication-free Tableaux Calculi Together with Cut-free and Contraction-free Sequent Calculi for the Interpolable Propositional Intermediate Logics. Logic J. IGPL 7, 447–480 (1999)
2. Avron, A.: Using Hypersequents in Proof Systems for Non-classical Logics. Annals of Mathematics and Artificial Intelligence 4, 225–248 (1991)
3. Avron, A.: Gentzen-Type Systems, Resolution and Tableaux. Journal of Automated Reasoning 10, 265–281 (1993)

4. Avron, A.: A Simple Proof of Completeness and Cut-admissibility for Propositional Gödel Logic. Journal of Logic and Computation (2009), doi:10.1093/logcom/exp055
5. Avron, A., Konikowska, B.: Decomposition Proof Systems for Gödel Logics. Studia Logica 69, 197–219 (2001)
6. Baaz, M., Ciabattoni, A., Fermüller, C.G.: Hypersequent Calculi for Gödel Logics - a Survey. Journal of Logic and Computation 13, 835–861 (2003)
7. Baaz, M., Preining, N., Zach, R.: First-order Gödel Logics. Annals of Pure and Applied Logic 147, 23–47 (2007)
8. Baaz, M., Zach, R.: Hypersequents and the Proof Theory of Intuitionistic Fuzzy Logic. In: Clote, P.G., Schwichtenberg, H. (eds.) CSL 2000. LNCS, vol. 1862, pp. 187–201. Springer, Heidelberg (2000)
9. Ciabattoni, A., Galatos, N., Terui, K.: From Axioms to Analytic Rules in Nonclassical Logics. In: Proceedings of LICS, pp. 229–240 (2008)
10. Corsi, G.: Semantic Trees for Dummett's Logic LC. Studia Logica 45, 199–206 (1986)
11. Dyckhoff, D.: A Deterministic Terminating Sequent Calculus for Gödel-Dummett Logic. Logic J. IGPL 7, 319–326 (1999)
12. Dyckhoff, D., Negri, S.: Decision Methods for Linearly Ordered Heyting Algebras. Archive for Mathematical Logic 45, 411–422 (2006)
13. Dummett, M.: A Propositional Calculus with a Denumerable matrix. Journal of Symbolic Logic 24, 96–107 (1959)
14. Gabbay, D.: Semantical Investigations in Heyting's Intuitionistic Logic. Reidel, Dordrechtz (1983)
15. Gödel, K.: On the Intuitionistic Propositional Calculus. In: Feferman, S., et al. (eds.) Collected Work, vol. 1, Oxford University Press, Oxford (1986)
16. Hájek, P.: Metamathematics of Fuzzy Logic. Kluwer Academic Publishers, Dordrecht (1998)
17. Metcalfe, G., Olivetti, N., Gabbay, D.: Proof Theory for Fuzzy Logics. Springer, Heidelberg (2009)
18. Sonobe, O.: A Gentzen-type Formulation of Some Intermediate Propositional Logics. Journal of Tsuda College 7, 7–14 (1975)
19. Takano, M.: P Another proof of the strong completeness of the intuitionistic fuzzy logic. Tsukuba J. Math. 11, 851–866 (1984)
20. Takeuti, G.: Proof Theory. North-Holland, Amsterdam (1975)
21. Takeuti, G., Titani, T.: Intuitionistic Fuzzy Logic and Intuitionistic Fuzzy Set Theory. Journal of Symbolic Logic 49, 851–866 (1984)

Author Index